

End-User Service Computing: Spreadsheets as a Service Composition Tool

Željko Obrenović and Dragan Gašević

Abstract—In this paper, we show how spreadsheets, an end-user development paradigm proven to be highly productive and simple to learn and use, can be used for complex service compositions. We identify the requirements for spreadsheet-based service composition and present our framework that implements these requirements. Our framework enables spreadsheets to send requests and retrieve results from various local and remote services. We show how our tools support different composition patterns and how the style of declarative dependencies of spreadsheets can facilitate service composition. We also discuss novel issues identified by using the framework in several projects and education.

Index Terms—Online information services end-user development and service-oriented computing, information systems applications office automation and spreadsheets, software engineering distribution, maintenance, and enhancement.

1 INTRODUCTION

END-USER development (EUD) is a very popular form of interaction with computers. Millions of users daily create their own solutions using spreadsheets, scripting, or high-level visual programming languages. It is estimated that, in 2005, in the US alone, there were 55 million end-user developers compared to 2.75 million professional software developers [5] and the number is growing [17]. Scaffidi et al. estimated that, in 2012, there will be 90 million computer end users in American workplaces, with more than 55 million of these using spreadsheets or databases [47].

Although their number is significant, end-user developers exploit only a very limited number of available software services. EUD environments are not connected to the world of service-oriented computing (SoC), which promises to bring thousands of novel and easily reusable software components [14], [28], [61]. Existing development environments for service-oriented solutions, on the other hand, are not appropriate for end users as they require expertise of professional developers.

Joining EUD and SoC paradigms in a more synergic mix can possibly enable millions of users to access numerous software services and components. The long-term goal of our work is exploring the potential of this mix, and our first step is exploring how existing EUD environments, with a well-established user base, can be adapted to support advanced service compositions.

In this paper, we describe how spreadsheet environments can facilitate complex service compositions. We show that, with only a few easy to learn extensions,

spreadsheets, a paradigm proven to be highly productive and simple to learn and use, can be employed for complex service compositions. Our results are based on the experiences in implementation and usage of a framework, called AMICO:CALC, which connects existing spreadsheet environments with a wide variety of software services. The framework defines a service coordination model where service composition is achieved through spreadsheet formulas, allowing users to combine diverse types of software services, such as connecting a Google search Web service to a local text-to-speech (TTS) service. For readers who would like to further explore presented topics, we provide an open source package that contains the source code of our platform and software components used in the examples presented in the text and the appendices.¹

We first introduce a motivating scenario that illustrates the type of EUD that we aim to support. We then present some of the existing solutions and discuss their limitations. After that, we identify the key requirements for spreadsheet-based service composition. We describe the implementation of these requirements within our AMICO:CALC framework for spreadsheet-based service composition. We separately describe the details of the middleware part of our framework and its integration into existing spreadsheet programs. We show how our spreadsheet EUD framework supports different service composition patterns and how the style of declarative dependencies of spreadsheets can facilitate service composition. We also discuss novel issues identified by applying the framework. Our supplemental materials, which can be found on the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/TSC.2008.16>, include three appendices that provide more detail on the types of software services supported by our environment, the adapters for service data structures, and a description of the implementation of the scenario introduction in Section 2 using our framework, as well as a video.

• Ž. Obrenović is with the User Centered Engineering Group, Department of Industrial Design, Technische Universiteit Eindhoven (TU/e), Eindhoven, The Netherlands. E-mail: z.obrenovic@tue.nl, obrenovic@acm.org.

• D. Gašević is with the School of Computing and Information Systems, Athabasca University, 1 University Drive, Athabasca, AB T9S 3A3, Canada.

E-mail: dragang@athabascau.ca, dgasevic@sfu.ca, dgasevic@acm.org.

Manuscript received 25 Feb. 2008; revised 20 June 2008; accepted 8 Dec. 2008; published online 17 Dec. 2008.

For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSC-2008-02-0014. Digital Object Identifier no. 10.1109/TSC.2008.16.

1. Available at <http://amico.sourceforge.net/amico-calc.html>.

2 MOTIVATING SCENARIO

In this section, we introduce the motivating scenario to clarify our definitions of end users and end-user service composition.² We define end-user service composition as the service composition performed by users who are not professional developers, that is, users whose main job is outside of computer science, in areas such as humanities, engineering, medicine, or business. These users are usually very sophisticated in the fields of their expertise, but they have little time or interest in learning a programming language or software engineering methodology, and they use service composition tools as a means to accomplish their main (professional or private) tasks. For a more elaborate view on end users, we recommend [6] and [16]. Here, we give a sample scenario that illustrates the type of end-user service composition that we are aiming at, with Michelle as our end-user developer.

Michelle is a writer, preparing a book about Dutch paintings. To collect necessary information, she analyzes different sources about Dutch culture, most of them in the Dutch language. However, being a beginner in Dutch, she often has to translate phrases from and to English. She also needs to find additional information about particular concepts and facts. Instead of using several tools, such as online dictionaries, definition books, and search engines, she decides to compose a simple service that aggregates all the services she needs.

She opens her spreadsheet program and, using a special extension, connects to software services, including an online translation service, Web search service, a spelling checker service, and local services such as TTS engines for English and Dutch, as well as a WordNet definition service, which gives her definitions of terms and lists of synonyms. She then composes a spreadsheet where, as a resulting functionality, she can type a word or phrase in Dutch and get various results for it, including spelling corrections, translation(s) in English, and additional information about the phrase, such as definitions from WordNet or a list of Web pages that contain data about the entered phrase. Although she could use these services individually, the possibility to quickly compose an aggregate service in a spreadsheet enables her to get a simple unified user interface, adapted to her needs, saving her time, so that she can focus on her main task. Yet, she is able to compose those services on her own without any additional assistance.

She also makes a version of the spreadsheet that is connected to her Web browser so that she can select the phrase in the Web page and, by using the browser toolbar, send it to the spreadsheet without the need to copy the text and switch between the windows. She adds a TTS service to her spreadsheet so that she can hear the translated phrase instead of switching windows to look at the result.

Finally, by changing just a few formulas, she makes a third version of the spreadsheet, which is connected to a short message service (SMS) service. When she sends an SMS to her virtual phone number, the spreadsheet, which

she left running on her machine before leaving the house, receives the message, translates the phrase, and sends her back an SMS message with the translated text and additional definitions. She uses this service when she is in a library or museum.

3 SPREADSHEETS AND SOFTWARE SERVICES

Our example scenario illustrates the need for simple yet powerful EUD environments and integrated use of various software services. Here, we describe some existing end-user service composition solutions and discuss their limitations for supporting applications described in the example scenario. We start with a brief discussion about spreadsheets, SoC, and service composition and then describe some spreadsheet service extensions.

3.1 Spreadsheets Basics

A spreadsheet is a computing application that displays a rectangular table (or grid) of information, consisting of *text* and *numbers*, where values sit in *cells*. Spreadsheets allow defining the type of data for each cell (usually limited to texts and numbers) and defining how different cells depend on each other. The relationships between cells are defined through *formulas*. Users can interactively change the data and formulas and immediately see the effects of their actions. By modifying the selected values, for example, the users can see how all the other values change accordingly. This enables studying of various what-if scenarios, which makes spreadsheets a good rapid-prototyping environment. At a more abstract level, it is often convenient to think of a spreadsheet as a *mathematical graph*, where the nodes are spreadsheet cells and the edges are references to other cells specified in formulas, which is often called the *dependency graph* of the spreadsheet. References between cells can take advantage of spatial concepts such as cell relative and absolute positions, as well as named locations, to make the spreadsheet formulas easier to understand and manage.

Many people find it easier to perform calculations in spreadsheets than to write the equivalent sequential program [11], [49]. One of the main properties of spreadsheets is its usage of human spatial perception and reasoning: *spreadsheets are designed to perform general computation tasks using spatial relationships rather than time as the primary organizing principle*. The ability to define a set of cells with a spatial relation to one another, exploiting users' natural spatial perception and reasoning, is one of the key properties for the success and wide use of spreadsheets. Many of the concepts common to sequential programming models have analogs in the spreadsheet world. For example, the sequential model of the indexed loop is usually represented as a table of cells with similar formulas (normally differing only in which cells they reference).

There are many commercially available spreadsheet environments. The spreadsheet paradigm, however, includes not only commercial spreadsheet systems such as MS Excel and OpenOffice.org CALC but also a number of research languages that extend the paradigm with features such as gestural formula specification [9], [33], graphical types [57], visual matrix manipulation [3], [55], high-quality

2. Appendix C, which can be found on the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/TSC.2008.16>, shows the implementation of this scenario with our solution.

visualizations of complex data [12], and specification of graphical user interfaces (GUIs) [35]. Most of the current research on spreadsheets is focused on reducing the errors introduced by users [7], [19], [53].

3.2 Service-Oriented Computing and Service Composition

Service computing provides higher level abstractions facilitating the implementation and configuration of software applications in a manner that improves productivity and application quality and enables organizing applications for large-scale open environments [28]. In SoC, services are usually defined as autonomous platform-independent computational elements that can be described, published, discovered, orchestrated, and programmed using standard protocols for the purpose of building networks of collaborating applications distributed within and across organizational boundaries. Web services are currently the most promising technology based on the idea of SoC [60]. SoC is, however, not limited to Web services but embodies key principles such as loose coupling, implementation neutrality, flexible configurability, persistence, granularity, and teams [28].

SoC involves various service layers, functionality, and roles [43]. Basic services, their descriptions, and basic operations (i.e., publication, discovery, selection, and binding) that produce or utilize such descriptions constitute the foundation of service-oriented architectures. The higher layers provide the additional support required for service composition and service management.

The need to aggregate or combine small services into larger services is the core in service-oriented architectures. The *service composition* is a process of aggregating multiple services into a single composite service. In many cases, a single service will act as a front end to many small services. There are a variety of methods for combining services, including simple pipes and filters, which direct the output of one service into the input of another service, and more complex choreographies and orchestrations, which utilize a high-level declarative and scripting language to control the sequence and flow of service execution [42]. Web Services Choreography Description Language (WS-CDL) is an example of a complex choreography language.

Most of the existing service compositions solutions are aimed for supporting professional developers, with solutions such as automated service composition, model-driven service composition [41], Semantic-Web-enabled composition [45], QoS-aware service composition [34], and business-driven automated composition [59].

Currently, end users appear mostly as the consumers of a final service composition, and they cannot influence the service composition in a similar manner as they build custom applications in end-user environments. Most existing service composition solutions do not enable end users to fully support their business process without needing to provide all the context information for automatic algorithms to generate a service composition for them.

3.3 Spreadsheets and Services

There are several solutions that provide a limited relation of SoC and spreadsheets. Most of them introduce predefined

functions that access online data about financial and similar services, but some of them introduce functions that enable the usage of other primarily Web services. In this section, we describe Web service extensions of commercial spreadsheets, online spreadsheets, and some task-specific spreadsheet service extensions.

3.3.1 Using Web Services within Spreadsheets

Commercially available spreadsheet environments have mechanisms to extend their functionality, enabling developers to build custom functions and extensions, some of which may access Web services. MS Excel's Web Service extension [32], for example, enables end users to create a code wrapper for a Web service and use functions from this wrapper within the spreadsheets. In this case, end users call a Web service through spreadsheet macrolanguage and define functions that return the results of a service. This approach, however, requires manual coding of every service connection and is limited to Web services. StrikeIron SOA Express for Excel [51] (previously known as OnDemand Web Services for Excel) provides a similar solution with a simpler drag-and-drop interface for the connection of Web service parameters and spreadsheet fields.

The basic idea behind these extensions is in making it possible to have "live" cells within spreadsheets, i.e., the cells that are updated with data taken from Web services, and they are less suitable for service composition.

3.3.2 Online Spreadsheets

Rich application development abilities of Web libraries enabled the creation of fully functional online spreadsheets.³ Online spreadsheets allow users to create a spreadsheet and have multiple persons edit and share it on the Web. NumSum [36], iRows [29], Zoho [62], EditGrid [15], and Simple Spreadsheets [50] are examples of such infrastructures. Some of these solutions enable developers to remotely access and modify data within spreadsheets.⁴ For example, Zoho introduces several APIs that enable developers to access spreadsheet data through REST and XML-RPC-based interfaces. EditGrid, on the other hand, enables access to spreadsheets through REST- and SOAP-based APIs and instantiation of EditGrid's spreadsheets as JavaScript objects. There are also several open source online spreadsheet infrastructures.

Within these examples, however, the emphasis of service extensions is on using spreadsheets as a data source by other applications and not on using spreadsheets as a service composition tool. Limited forms of service composition within spreadsheets can be found in more advanced online spreadsheets such as Google spreadsheets [25]. For this purpose, Google spreadsheets introduce two new functions: GoogleFinance and GoogleLookup. GoogleFinance enables end users to use various financial services such as getting stock prices and currency exchange rates. GoogleLookup provides results of a search service about "people, places, and things." Google Web spreadsheets, however, do not directly support access to other services, i.e., they enable the usage of a limited number of predefined financial and Web search services. Moreover, these solutions cannot be used for work with other types of services, while we aim at supporting service composition with various types of

3. http://en.wikipedia.org/wiki/List_of_online_spreadsheets.

4. See www.programmableweb.com/apitag/spreadsheet for details.

service interfaces and technologies (see Section 4.1 and Appendix A, which can be found on the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/TSC.2008.16>, for details). Functionality similar to Google spreadsheets is provided by wikiCalc [56]. In addition to standard spreadsheet functions, users can use the `wkcHTTP()` function, which gives wikiCalc access to external Web services, accessing a specific URL on the Internet using the HTTP protocol. EditGrid and iRows also support live stock and currency data functions but not access to other services.

3.3.3 Task-Specific Spreadsheet Service Extensions

Kandogan et al. developed A1, a spreadsheet-based environment with a task-specific system-administration language [31]. A1 extends the spreadsheet paradigm by introducing objects in cells so that cells can contain an arbitrary Java object. In this way, A1 can access many of the system services using Java service libraries. System administrators can use this system to access remote systems, gather status data, and orchestrate control of disparate systems in a uniform way. A1, however, is optimized for system administrators and requires knowledge of the Java programming language, i.e., it is not suitable for most of the end users.

To support some of the tasks related to online forms, Fujima et al. developed a C3Sheets prototype [22], enabling end users to create custom interfaces on top of existing Web applications and services using a spreadsheet-like tool. Their tool makes possible clipping of input and result elements from existing services to form cells on a spreadsheet, connecting these cells using formulas, and cloning cells, so that multiple requests can be handled side by side. The tool, however, is not a fully functional spreadsheet environment, and it is built to support work with standard HTML pages but not for other services such as SOAP Web services.

4 REQUIREMENTS FOR SPREADSHEET-BASED SERVICE COMPOSITION

Spreadsheets usually provide fixed functionality with limited service extensions and do not include support for diverse services and service compositions needed for implementation of the functionality described in our example scenario. In this section, we identify basic requirements for going beyond existing SoC and spreadsheet solutions, providing a more synergic mix of these two paradigms, extending the spreadsheet paradigm into a service coordination language.

Our main goal is to enable spreadsheet-based service composition, where coordination among diverse services is achieved by end users through spreadsheet formulas. We want, therefore, to extend spreadsheets into a fully functional language for service coordination. To state our goal in a more structured way, we followed a constructive approach to the definition of coordination languages described in [13], which consists of identifying the following components:

1. *Coordination entities.* These are the entity types that are coordinated. These could be Web services, Unix-like processes, threads, concurrent objects, and even users.

2. *Coordination media.* These are the media making communication among the entities possible. Moreover, a coordination medium can serve to aggregate entities that should be manipulated as a whole. Examples are classic media such as semaphores, monitors, or channels or more complex media such as tuple spaces, blackboards, or pipelines.
3. *Coordination laws.* A coordination model should dictate a number of laws to describe how entities coordinate themselves through the given coordination media and using a number of coordination primitives. Examples are laws that enact either synchronous or asynchronous behaviors or exploit explicit or implicit naming schemes for coordination entities.

4.1 Requirements for Coordination Entities

Our motivating scenario illustrates the need for diverse software services, and we want to enable end users to *access a wide variety of software services*. Our goal is not only to support one type of coordination entities, such as Web services, but any remote or local service (such as speech input or output) through a diverse set of communication interfaces such as XML-RPC, OpenSound Control, or simple TCP and UDP interfaces (see Appendix A, which can be found on the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/TSC.2008.16>, for details). We are therefore looking at a more generic definition of a service as a self-contained functional unit in which service consumers interact with the service through a well-defined interface. The solutions should be open to a wide range of such service interfaces.

4.2 Requirement for Coordination Media

The coordination media should serve as glue between diverse software service interfaces and spreadsheets, being able to do the following:

- *Abstract the difference among service interfaces.* Finding an appropriate abstraction is crucial if we want to support a broad interconnectivity of services and support our requirements for the work with diverse entities.
- *Provide simple data structures that are easy to map to spreadsheet cells.* Our goal is to make spreadsheet extensions simple so that existing environments can be more easily adapted. When the coordination media abstracts and simplifies data structures, integrating services into existing environments is generally simpler.

Coordination media should facilitate solving one of the key problems in enabling end users to use heterogeneous software services, that is, working with significantly different data abstractions that software services and spreadsheets use. Web services, for example, operate with complex hierarchical XML data structures, and end users are usually not able to easily understand and use such structures.

4.3 Requirements for Coordination Laws

The composition primitives used in spreadsheet formulas should be complex enough to enable supporting the

functions described in our motivating scenario, including the following:

- *Providing flexible and generic compositions primitives.* That is, we do not limit composition to a particular task.
- *Supporting basic communication and workflow patterns such as sequence, parallel split, exclusive choice, simple merge, and synchronization* [58]. With these patterns, it is possible to create various complex service composition graphs. Support for more complex patterns such as structured synchronizing merge or blocking partial join is optional, as, in our experiences, these complex patterns are hard to understand and manage by end users.
- *Supporting mappings between spreadsheet spatial relationships and service data with temporal dimensions.* In other words, we want to avoid the need for users to write complex adapters for services and to directly map spreadsheet services with service parameters and results.

Our goal, however, is not to create a complex service-oriented architecture but to enable users to directly combine software services through simple spreadsheet formulas. The additional requirement for our coordination laws is that they should be *understandable to end users and easy to integrate within existing spreadsheet environments*. Although the dividing line between end users and professional developers seems to have become blurred, we wish to emphasize that our notion of end users is quite strict. We assume that an end user should not be able to do any scripting beyond simple formulas and an occasional conditional expression by using the if-then construct. We also do not want to propose new interaction techniques, as building new development environments is a tedious and time-consuming task and it is hard to predict how users will accept it. Adapting existing spreadsheet environments enables users to build on previous experience and learn faster and more efficiently how to compose services.

5 A FRAMEWORK FOR SPREADSHEET-BASED SERVICE COMPOSITION

By following the requirements defined in Section 4, we have developed AMICO:CALC, a framework for end-user spreadsheet-based service composition. Our framework is an extension of the Adaptable Multi-Interface COmmunicator (AMICO) middleware platform,⁵ which facilitates adaptation, abstraction, and mediation of diverse software services [37]. The AMICO:CALC framework uses a simpler version of AMICO, with the addition of spreadsheet add-ons, connecting existing spreadsheet environments with services and extending them with service composition functions.

AMICO:CALC, as a whole, facilitates the loosely coupled integration of heterogeneous software services within spreadsheet environments while providing a uniform and simple interface to end users. It consists of two main parts (Fig. 1):

- *Middleware for heterogeneous services*, with support for diverse service interfaces. The middleware supports our coordination entity and coordination media requirements.

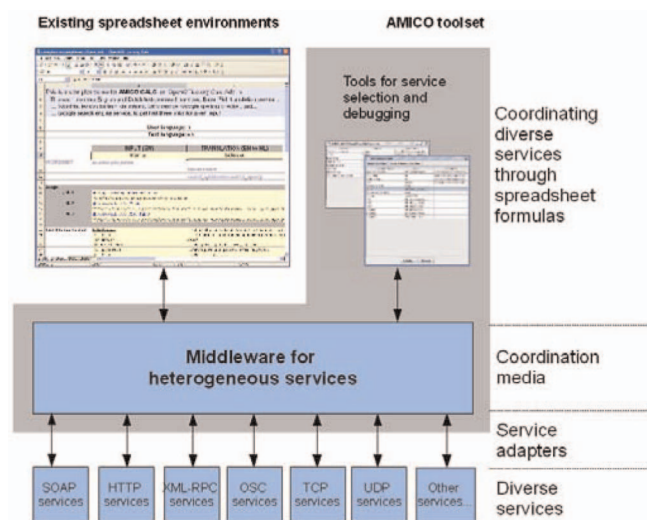


Fig. 1. A framework for end-user spreadsheet-based service composition. End users use standard spreadsheet environments (with a specific add-on) to access the middleware for heterogeneous software services. Our AMICO middleware introduces simple data structures and data adapters for diverse service interfaces. The framework also offers the tools for service selection and debugging.

- *Spreadsheet extensions for service composition*, which introduces several novel functions for service composition through spreadsheet formulas. We have introduced only a few simple functions and implemented defined functions within the OpenOffice.org CALC spreadsheet environment. The add-on supports our coordination law requirements.

The middleware maintains a list of *variables* (see Section 6 for details) which encapsulate data structures used by services, and our spreadsheet extensions map these variables to spreadsheet formulas (see Section 7 for details). Due to the many limitations of the spreadsheet extension mechanism and keeping in mind that our middleware uses components implemented in diverse programming languages, we were not able to embed the full functionality of the middleware within the spreadsheets. Therefore, spreadsheet and middleware environments are connected in a loosely coupled way, each running in a separate process and communicating through TCP and UDP connections. This approach makes spreadsheet extensions simpler and easier to implement as they only read and write variables, while our middleware maps variables to service interfaces. One of the consequences of this approach is that service data structures, represented in variables, are not immediately visible within spreadsheets, i.e., it is necessary to write formulas to get variables within the spreadsheets. We do, however, provide a mechanism that enable a user to import all variables with one function or copy them from the middleware control panel and put them on a different page or tab of the spreadsheet, thus making available services more exposed to the user.

In comparison with the existing solutions, AMICO:CALC has a wider scope than most of existing solutions, as it can work with diverse types of software services and it does not only support data-aggregation but also integration of additional services such as interaction facilities involving speech recognition and TTS output. Table 1 gives a comparison of AMICO:CALC to some of the existing solutions in terms of their flexibility, types of services supported, and main limitations.

5. <http://amico.sourceforge.net>.

TABLE 1
Comparison of Some of the Existing Spreadsheet Service Solutions with AMICO:CALC

Solution	Task	Using data from services within spreadsheet formulas	Outputting spreadsheet data through a service interface	Main limitations
Excel's Web Services extension	Any	Using data from Web SOAP services	Not supported	Limited to Web services, requires some programming
StrikeIron SOA Express for Excel	Any	Using data from Web SOAP services	Not supported	Limited to Web services
Google spreadsheets	Any	Predefined functions for access to financial and Web search data (GoogleFinace and GoogleLookup functions)	Not supported	Limited to predefined number of financial and search services
wikiCalc	Any	wkcHTTP() function for access to Web pages	Not supported	Limited to HTTP GET/POST services
NumSum, Zoho, and EditGrid, iRow	Any	Predefined functions for access to live stock and currency data	Applications can access spreadsheet data through REST, SOAP, or XML-RPC based APIs	Limited to predefined number of financial and search services
A1	System administration	Access to data from system services and Java objects	Not supported	Application specific, requires knowledge of Java and new syntax
C3Sheets	Clip, connect, clone	Using (HTML) input from Web pages	Not supported	Task specific, lack of standard advanced functions of spreadsheets
AMICO:CALC	Any	Introduce functions (described in Section 7) to, both, receive data from and to send data to any of the services supported by the AMICO middleware, including SOAP Web services, legacy HTTP GET and POST services, XML-RPC, OSC, and simple TCP, UDP services (see Section 6.4 and Appendix A for details). Services connected to the AMICO middleware can also read data updated within the spreadsheet.		Lack of tools that can facilitate end-users to debug AMICO services (see Section 6.4 for details)

In the following two sections, we describe details of AMICO:CALC by describing first its support for coordination entities and media (Section 6) and then its support for coordination laws (Section 7) as specified in the requirements in Section 4.

6 COORDINATION ENTITIES AND MEDIA: ADAPTING AMICO MIDDLEWARE

As a middleware for our solutions, we have adapted the service brokering infrastructure, called AMICO. Adaptation included simplifying the service adapters and the addition of tools that can make easier work with services for end users. AMICO is a generic platform, realized as a Java application, used to support rapid prototyping with heterogeneous software services [37]. The adaptation of AMICO, included in AMICO:CALC, is a complex toolset, which includes the following:

- *AMICO service brokering infrastructure* (coordination media). This includes a shared data space for simple data structures and a notification service.
- *Set of AMICO service interfaces* (interfaces toward heterogeneous coordination entities). This enables

the use of services with diverse interfaces, supporting our requirement for the integration of a wide variety of software services.

- *AMICO service data adapters*. These map heterogeneous service data structures to a shared data space and linearize complex data structures to simpler data structures suitable for use within spreadsheets (see Appendix B, which can be found on the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/TSC.2008.16>, for details). In this way, we support our coordination media requirements for abstracting the difference among service interfaces and providing simple data structures.
- *AMICO tools for service selection and debugging*. These facilitate end users' work with heterogeneous services.

In the following sections, we briefly describe each of these elements.

6.1 Coordination Media: AMICO Service Brokering Infrastructure

AMICO is a blackboard type of coordination media, based on the publish-subscribe design pattern [23], which is well

suited for the integration of loosely coupled components, and often used in context-aware and collaborative computing [54]. In this model, a publisher updates a shared data repository without being concerned with whether any subscribers are listening for updates. In such a loosely coupled model, components can run on different machines in a distributed environment.

As a main data abstraction, AMICO provides a set of *untyped variables (named slots)* similar to untyped data structures used in end-user spreadsheet environments. Applications communicate by exchanging events through a shared data repository consisting of these variables. An application can update the variables and register for notifications about changes of any variable. The simplicity of data structures is the key property of AMICO that makes it easy for integration with spreadsheets.

In its basic ideas, AMICO is similar to other loosely coupled and notification architectures such as Elvin [20], Lotus Placeholder, which is based on the Notification Service Transfer Protocol [44], and tuplespace systems, such as Linda [24], Stanford EventHeap [30], and JavaSpaces [18].

6.2 Supporting Diverse Coordination Entities: AMICO Service Interfaces

One of the key differences between AMICO and other loosely coupled services is support for more than one integration interface. AMICO provides a unified view on different service interfaces and interconnects them in the common space. We support several widely used and standard communication protocols that various software services use, such as low-level TCP and UDP interfaces, Bluetooth, and many higher level interfaces such as HTTP GET/POST, XML-RPC, OSC, SOAP, and many application-specific interfaces.

AMICO is extensible, and it is possible to add new communication interfaces. This ability allows using not only Web services via the AMICO SOAP interface but also any local or remote software service or process that opens any of the supported open communication interfaces.

Appendix A, which can be found on the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/TSC.2008.16>, elaborate on used service interfaces and types of services that can be used through these interfaces.

6.3 Unifying and Simplifying Service Data Structures: AMICO Service Data Adapters

AMICO service data adapters map AMICO variables to the arguments of service calls and results of services and define the conditions when the service will be called.

For each of the communication interfaces, it is possible to define a service adapter (see Appendix B, which can be found on the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/TSC.2008.16>, for details on these adapters). An adapter defines the mapping between variables and service calls and between variables and results of services.

If the result is a complex XML structure, which is often the case with SOAP services, it is possible to simplify this data structure by providing XPath expressions to extract a part of the XML and, optionally, an XSLT transformation script if a more complex transformation is required. The

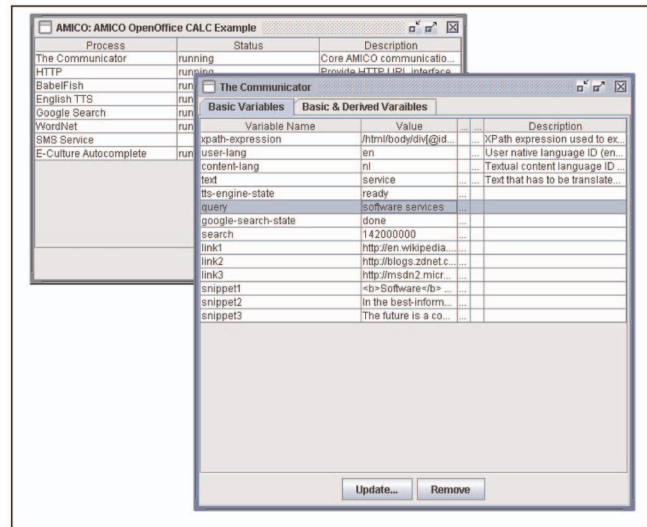


Fig. 2. Control panel of our middleware that end users can use to start services and check and change service variables.

basic idea of service adapters is to linearize service data structures to a set of variables, which can be easily mapped to spreadsheet cells and formulas, so that an end user does not have to deal with complex hierarchical structures. When complex data structures are used, more than one variable is usually derived. For example, if the result of a service is a list, such as a list of links to Websites from the Google search service, then we derive several variables, each for one element of the list. With very complex structures, it is sometimes better to define several adapters, each using just part of the data structures to variables, mapping only the elements that end users need to for a particular task.

6.4 AMICO Tools for Service Selection and Debugging

AMICO facilitates a loosely coupled connection of services, where each service runs as an independent process on the same or a remote machine. In order to make the launching of services and service adapters easier for end users, we provide several tools.

The control panel of our middleware, for example, enables an end user to select a service from a list and create a group of services (Fig. 2). The end user can then save the group and select it next time to automatically run all the processes. We also provide an interface where all currently run services can export the variables they use and the end user can read the description of these variables and change their values in order to check how they affect other variables. The variables shown in a window are automatically generated from a service adapter or updates by services, which makes developers' work of defining service adapters easier, as they do not have to provide lots of additional documentation to make their service adapter understandable to end users. This tool is also useful for debugging, as it allows for the monitoring of all the variables that are exchanged among services.

One of the limitations of the current implementation of AMICO:CALC is a lack of tools that can enable end users to define service adapter mapping without any help from developers. However, the focus of this paper is on using services within the spreadsheets when these adapters are

defined. Building intuitive service adapter configuration tools is, however, a part of our ongoing work.

7 COORDINATING SERVICES THROUGH SPREADSHEET FORMULAS

We have implemented the spreadsheet extension for OpenOffice.org CALC by using OpenOffice.org SDK 2.2.0 (the add-on can be used for OpenOffice.org 2.0.4 or newer). As our middleware introduces simple data structures and hides and abstracts most of the complexity of service interfaces and data structures, the implementation of the spreadsheet functions was straightforward and did not require a lot of code.⁶ We also provide a less functional Microsoft Excel add-in, which we do not discuss in this paper, as it is still a work in progress.

We introduced four new functions as a part of our spreadsheet extension (Table 2), in order to support our requirement for coordinating diverse services through spreadsheet formulas (see Section 4). After users select and start the services within the middleware, they can then use these four functions to make various service compositions. Due to the limited space, in this section, we show simple examples, while Appendix C, which can be found on the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/TSC.2008.16>, illustrates how these functions have been used to implement the motivating scenario described in Section 2, where we have used several Web services in combination with TTS, Wordnet, and other heterogeneous services and integrated these services with a Web browser and an SMS.⁷

In the next two sections, we describe how these functions, in combination with the built-in spreadsheet functionality, support our requirements for coordination laws defined in Section 4.3.

7.1 Supporting Basic Composition Patterns

As basic primitives for service composition in spreadsheets, we introduce two simple functions for asynchronous reading and updating of AMICO variables: `AMICO_WRITE` and `AMICO_READ`.

The introduced functions directly reflect the basic interface of the AMICO infrastructure, that is, updating variables and receiving notifications about variable updates. For example, the expression `AMICO_WRITE("spelling"; B10)` calls the Google spelling checker service (every time cell B10 is updated) and the expression `AMICO_READ("spelling-suggestion")` receives a value when the spelling service finishes. These two functions also resemble the basic spreadsheet cells' update-refresh mechanism.

With these two functions, it is possible to support basic communication patterns (and their combinations), including one-to-one, one-to-many, and many-to-one bindings. Fig. 3 illustrates how these functions can be used to support workflow patterns *sequence*, *parallel split*, *exclusive choice*, and *multimerge*. `AMICO_WRITE_DELAYED`, described in the

6. Available at <http://amico.sourceforge.net/>.

7. See also <http://www.idemployee.id.tue.nl/z.obrenovic/media/amico-calc-demo.avi>.

TABLE 2
Spreadsheet Extensions with Functions that Address Requirements for the Coordination Laws Defined in Section 4.3

Function
<p>AMICO_WRITE (<variable-names>, <values>)</p> <p>Updates AMICO variables with given values (returns the same value, or the last value from the range if a cell range is given).</p>
<p>AMICO_READ (<variable-name>)</p> <p>Registers for notifications about an AMICO variable with a given name and updates the spreadsheet every time when the value is received. Our spreadsheet extension creates a thread that listens for notifications for registered variables, and propagates the received values to spreadsheet formulas that use this value.</p>
<p>AMICO_WRITE_DELAYED (<variable-names>, <values>, <delays>)</p> <p>Updates AMICO variable(s) with a given values, after (a) given delay(s). For example, <code>AMICO_WRITE_DELAYED("A", "test", 2.5)</code> will update the variable A with the value "test" after 2.5 seconds. <code>AMICO_WRITE_DELAYED(A1:A10, B1:B10, C1:C10)</code> will update variables with names defined in cells A1 to A10, with values defined in cells B1 to B10, with delays defined in cells C1 to C10 (i.e., the function will first wait for a period defined in C1 and then update a variable with the name defined in cell A1 with the value from cell B1, then it will wait for a period defined in cell C2 before updating variables with the name defined in cell B2, and so on). The function returns the values as they are updated (in our last example, it returns B1, B2 ... B10).</p>
<p>AMICO_READ_LOOP (<variable>, <cell-or-row-id>, <start-value>, <end-value>, <step>)</p> <p>Maps sequential updates of an AMICO variable into a spatial update of spreadsheet cells. Updates are performed incrementally within a given row or column, with a given step. For example, <code>AMICO_READ_LOOP("var1", "column", "A", 5, 10, 1)</code>, will map ten updated of variable var1; the first update will update the cell A5, the second update to A6, and last the cell A14.</p>

following section, partially enables *synchronization* and *synchronization merge* workflow patterns.

`AMICO_WRITE` is also the main mechanism for a spreadsheet to export its results, as variables updated by `AMICO_WRITE` can be used by any other services connected to AMICO.

7.2 Mapping Spreadsheet Spatial Relationships to Service Data and Temporal Dimensions


The `AMICO_READ` and `AMICO_WRITE` functions can be used in any of the spreadsheet formulas in any of the cells. In this way, an end user can spatially organize the input and output of services, building a more intuitive interface where these two functions provide an effective means to map service spatial relations to service data structures. `AMICO_WRITE` can also receive a range of cells as arguments, enabling the mapping of multiple spreadsheet cells to multiple updates of AMICO variables by using only one function.

Sequence	A	B	C	D	E
	1	=AMICO_WRITE("tts-input",AMICO_READ("spelling-suggestion"))			
Sequence	A		B		
	1	=AMICO_READ("spelling-suggestion")			
	2	="Did you mean " & A1			
Parallel Split	A		B		
	1	=AMICO_READ("text")			
	2	=AMICO_WRITE("google-query",A1)			
	3	=AMICO_WRITE("spelling",A1)			
Exclusive choice	A		B		
	1	=AMICO_READ("spelling-suggestion")			
	2	="Did you mean " & A1			
Simple merge	A		B		
	1	=AMICO_READ("link1")			
	2	=AMICO_READ("wordnet-definition")			
	3	=AMICO_READ("translated-text")			
	4	="Link: " & A1 & ", def.: " & A2 & " (" & A3 & ")"			
5	=AMICO_WRITE("sms-message",A4)				

Fig. 3. Simplified spreadsheet formulas for supporting various workflow patterns with AMICO_READ and AMICO_WRITE functions.

Two additional functions, AMICO_WRITE_DELAYED and AMICO_READ_LOOP, provide means for mapping spreadsheet spatial dimensions to service temporal dimensions. AMICO_WRITE_DELAYED receives a range of cells and transforms the spatial ordering of cells into the sequential time flow of updates of AMICO variables. Fig. 4 illustrates how this function can be used with a MIDI player service to create a simple melody. The MIDI player is a simple service that can be controlled through a TCP interface, enabling other services to play a musical note if they update the

	A	B	C	D	E	F
1	Note	Velocity	Duration (ms)	Variable	Value	Pause (s)
2	71	100	400	midi-note	=A2 & "" & B2 & "" & C2	=C2/1000
3	71	100	400	midi-note	71 100 400	0.4
4	71	100	400	midi-note	71 100 400	0.4
5				pause		0.5
6	71	100	400	midi-note	71 100 400	0.4
7	71	100	400	midi-note	71 100 400	0.4
8	71	100	400	midi-note	71 100 400	0.4
9				pause		0.5
10	71	100	400	midi-note	71 100 400	0.4
11	73	100	400	midi-note	73 100 400	0.4
12	69	100	400	midi-note	69 100 400	0.4
13	70	100	400	midi-note	70 100 400	0.4
14	71	100	400	midi-note	71 100 400	0.4
15				pause		0.5
16	=AMICO_WRITE_DELAYED(D2:D15; E2:E15; F2:F15)					



MIDI service (with TCP connection to AMICO)

Fig. 4. Illustration of the work of AMICO_WRITE_DELAYED. AMICO_WRITE_DELAYED sends notes to the MIDI service, in this case, a variation of the Jingle Bells song. AMICO_WRITE_DELAYED receives the list of variables that should be updated (column D), the list of values that will be used to update the variables (column E), and timing as a list of pauses between updates (column F).

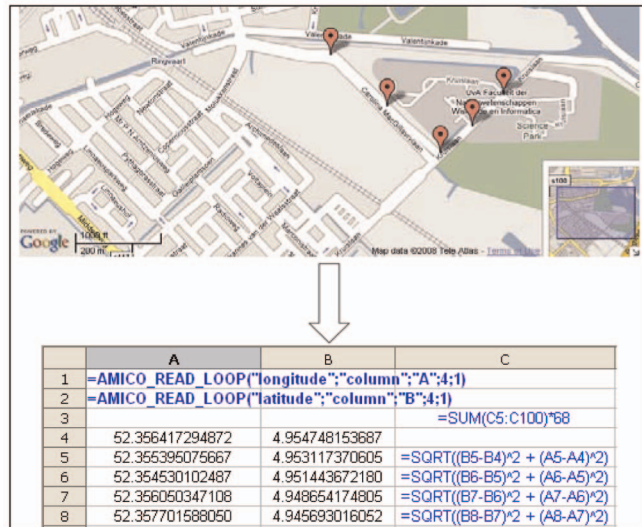


Fig. 5. An illustration of the work of the AMICO_READ_LOOP functions. This example maps a selection of coordinates from Google Maps to the updates of spreadsheet cells, and these updates are used to calculate the distance between the points.

AMICO variable "midi-note" with the value of the note, its velocity (intensity), and its duration.

AMICO_WRITE_DELAYED also solves some of the problems concerning temporal management. For example, in the implementation of our example scenario (Appendix C, which can be found on the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/TSC.2008.16>), we used an online text translation service in combination with two local TTS services. When a user requests a translation from Dutch to English, for example, we send the text to the translation service and to the Dutch TTS service, while the result of the translation service is sent to the English TTS engine. Because of the short delay, both TTS services produced the speech almost at the same time. Another related problem appears when AMICO_WRITE uses formulas depending on more than one cell. In this case, the function is called every time any depended cell is updated. This is problematic if the function has side effects such as sending SMS messages where it is desired to send the message only when all the variables that define the message are ready. The AMICO_WRITE_DELAYED function delays the update of the variables for a given number of seconds, solving the first problem. If AMICO_WRITE_DELAYED is called before the delay has passed, the timer is reset, the old value is ignored, and the new value will be sent, which solves the second problem; that is, the function "waits" until all relevant cells are updated and then sends the message. In this way, the function supports synchronization and synchronization merge workflow patterns.

AMICO_READ_LOOP introduces essentially the opposite effect to AMICO_WRITE_DELAYED: It transfers the sequential updates of one AMICO variable to spatial updates of spreadsheet cells. Fig. 5 illustrates how this function works on the example of the Google Maps coordinate selector. The Google Maps coordinate selector is an AJAX-based application that connects to AMICO through HTTP interfaces and updates the AMICO variables "longitude" and "latitude"

with the geographic coordinates of the location where the end user clicks on the map (also adding a marker on the map). `AMICO_READ_LOOP`, in this example, maps every update of variables to sequential updates of columns A and B, starting with row 4 (i.e., the first update will update cells A4 and B4, the second will update A5 and B5, and so on). Other spatial spreadsheet functions such as `SUM` can then be used to calculate other parameters such as the distance between the selected coordinates.

8 DISCUSSION

AMICO:CALC supports all requirements for spreadsheet-based software service composition, as specified in Section 4 and explained in Sections 5, 6, and 7. It enables the use of very diverse software services. The AMICO:CALC middleware introduces service interfaces that address this diversity and provides simple data structures that are easy to map to data structures used in spreadsheets. The introduced spreadsheet functions can support many complex service composition patterns and the mapping between spreadsheet spatial dimensions to service data structures and temporal dimensions. Our implementation of the OpenOffice.org CALC extensions is fully functional and illustrates how existing environments can be adapted to support service composition.

We have applied AMICO:CALC in two European projects, *Passepartout*⁸ and *K-Space*,⁹ where we used it as a tool for merging software components and services from various partners in several integrated demonstrations. We have also introduced AMICO:CALC in the course of Intelligent Multimedia Technologies at the Free University in Amsterdam, where students could prototype various interactive solutions with it. The course was realized with 32 undergraduate students (who are in their third or fourth year) from various departments, including cognitive systems, information systems, computer sciences, and artificial intelligence, and several exchange students [38]. We are currently using our framework in the Department of Industrial Design, Technische Universiteit Eindhoven, with designers and students using it in their projects, as well as in the undergraduate course on Sketching Interactive Systems. The main motivation for using AMICO:CALC in education was to enable students without programming experience to build complex interactive systems, as well as to enable quick “sketching” of various interactive systems without intensive programming.

In this section, we discuss some lessons learned, including novel user issues, performance issues, implications for spreadsheet designers, the potential for new applications, and possibilities to use other middlewares and EUD environments, in relation with conventional service composition solutions.

8.1 User Issues

The introduced spreadsheet functions proved to be easy to understand for most end users and powerful enough to enable usage of most of the complex services.

AMICO:CALC proved to be a very good tool for rapid prototyping as it allowed easy and real-time service compositions and demonstrations by inexperienced developers and end users. It is important to note that, although we clearly identified some of the usability problems such as the complicated installation procedure of middleware and the lack of tools that can facilitate end users to discover and install AMICO services (see Section 6.4 for details), our primary goal was to identify novel issues and possibilities. Therefore, our decision was to verify the feasibility of the proposed idea in real-world situations, rather than to test its usability in controlled conditions [26]. However, further usability testing is necessary if the solution is to be turned into the final (commercial) product.

Based on our experiences from the projects and the feedback from the students, we discuss some of the observed differences of our solution in comparison to ordinary spreadsheet functions.

8.1.1 Feedback about the State of Services

Existing spreadsheet functions have a relatively short response time, and if the number of formulas is not excessive, the response is usually perceived as immediate. In case of the usage of software services, however, the response time can be significantly longer, sometimes a few seconds, due to various factors such as network latency or performance overhead. If the services are chained (e.g., when we are sending results of a translation service to a TTS service), the response time for final results is even longer. For this kind of service, we found that it is very important to provide feedback about the state of the service. After the first tests with end users, we decided to change all AMICO service adapters so that they also update variables that describe the current state of the services (e.g., ready, working, and finished) and, thus, users can use this variable to make the state of the service visible.

The feedback about the service state also proved to be important in *debugging*. Without the feedback about the service state, especially within the complex spreadsheets, it is hard to see if the service is down or if there are some problems with the formulas.

8.1.2 Names of Variables

Although the introduced spreadsheet functions are simple, the names of variables used in these functions play a significant role in user understanding and usage of services. Our initial naming of variables was not intuitive enough for the end users in our experiment. For example, instead of calling a TTS service with the function `AMICO_WRITE("message-en"; <text>)`, the end users suggested that the function `AMICO_WRITE("say-it-in-english"; <text>)` is much more intuitive.

It is therefore very important to provide a careful mapping between service adapters and variable names and to involve end users in this process. Automatically generating service adapters, from WSDL descriptions, for example, usually produces names that are not intuitive for end users. We are also working on tools that can enable end users to define or change the names of the variables and define aliases.

8. <http://www.passepartout-project.org/>.

9. <http://kspace.qmul.net/>.

8.1.3 Using Online Services

While using online services, we experienced problems related to the registration and payment.

Registration. Many of the online services, such as Google, Yahoo, or Flickr Web services, required registration for every end user. These services use diverse methods for registration and sometimes require payments, having different business models (e.g., free for personal use, paying per request, or paying for a limited period). One of the open issues is how to simplify and unify the registration procedure for end users, that is, how to enable users to easily manage a huge number of passwords and keys that some of the services require and, yet, preserve the security and privacy of end users and their private data. A promising solution to this problem could be user-centered identity management solutions such as OpenID [40], as well as tools such as Sxipper [52].

Paying for services. A related issue concerns the provision of feedback about the price (money to be sent) for pay services. The end user's trust in the system and transparency of service calls plays an important role when the services are not free. Some users provided concerns about using services that they had to pay per request and indicated that they would like to have some kind of feedback about the current credit and the possibility to define a limit. Users do not feel comfortable when every spreadsheet update charges their credit card even for a small amount. An error in code can also be very expensive. For example, the Esendex SMS service charges about 0.10 per message. In one of the examples, we used this service within a complex spreadsheet, and we introduced an error causing an infinite refresh loop. In a few seconds, before we noticed the error, dozens of messages were sent. Left unnoticed, the spreadsheet could send thousands of messages in less than an hour, costing us hundreds of dollars and leaving the receiver of the messages busy with deleting the messages. This problem also influenced the design of service adapters and the AMICO service brokering infrastructure. To avoid calling the services every time service variables are updated, the adapters can define if they should be called when the variable is updated but not changed from the previous update. Such simple mechanisms may significantly reduce the number of service calls and response time, while avoiding unnecessary payments for services.

8.2 Performance Issues

Service composition using our framework is affected by the time overhead of the spreadsheet environment and AMICO middleware.

The spreadsheet environment introduces a delay that is a consequence of calculating formulas and cell dependencies, as well as a communication overhead between the AMICO middleware and the spreadsheet extension. Spreadsheets attempt to automatically update cells when the cells on which they depend have been changed, but it is hard to predict the exact order and time of cell value propagations, which limits the dynamic effect and temporal resolution of service composition.

The AMICO middleware, on the other hand, uses service adapters that do not support real-time interaction. AMICO

is therefore not suited for application with hard real-time requirements, as it makes no guarantees regarding the speed with which data will be transmitted to the recipients.

However, the delay introduced by the middleware processing and service adapters is usually of order of magnitude of 10 ms and was acceptable for most of our interactive applications [1]. To improve scalability, several instances of AMICO can be distributed in the network and interconnected.

8.3 Implications for Spreadsheet Designers

We experienced some problems due to the optimization techniques that spreadsheets use. Spreadsheets normally use functions without side effects; that is, the only result of spreadsheet functions is the value they return based on the input from other cells. Our functions, however, produce many side effects, such as calling the TTS engine or receive results from outside processes such as receiving the results of the Web service. Spreadsheet optimization techniques, however, do not account for such side effects. OpenOffice.org CALC, for example, sometimes does not evaluate formulas that do not affect the visible space. Microsoft Excel saves computed values when a user closes the document and reuses these values when the user again opens the spreadsheet while formulas are not evaluated.

Although these problems occur in a small number of cases and there are "tricks" that can be used to avoid them, such as making the results of a spreadsheet always affect the visible space, we would like to encourage spreadsheet designers to make these optimization techniques configurable and make their usage optional.

8.4 New Applications: Beyond Calculations

In addition to the extension of existing (primarily business-oriented) spreadsheets with additional data, our approach to using software services also opens possibilities for creating significantly different applications. The input for spreadsheets can now, for example, be detected by speech recognizer services, real-time facial expression detectors [39], or hand gesture detection [27]. The spreadsheet itself can produce many side effects such as calling Web services, generating speech output, or sending SMS messages. For example, by using only a few formulas, we were able to connect the Flickr service [21] with a simple local Web camera capture service to automatically upload captured images on the Web.

AMICO:CALC opens numerous possibilities for the exploration of heterogeneous open source components that are typically not used in EUD environments. The key to enabling novel applications lies in the ability of the AMICO platform to exploit a huge number of heterogeneous components and services [37]. When OSS components offer their functionality through any of the open communication interfaces, integration with AMICO:CALC is straightforward and consists of defining service adapters. With our open source distribution, we also provide adapters for widely used OSS components (e.g., SOAP, HTTP, and TCP). To enable the exploitation of their code by end users, we would like to encourage the OSS developers to include such stand-alone service-oriented examples with their distributions.

It is important to note that we see the role of AMICO:CALC here not as a final product environment but as an interface that can enable quick “sketching” of various software service combinations, such as sketching user interfaces [4]. In other words, our spreadsheets can enable quick exploration of possibilities of available software services and components and serve as an environment for studying of various what-if scenarios. In this manner, we used AMICO:CALC in domains such as rapid application development of multimodal interfaces, user interaction with the Semantic Web, and user coordination of devices in an ambient intelligent environment.

8.5 Toward General Requirements for End-User Service Computing: Using Other Middlewares and EUD Environments

Although the proposed framework is based on the extension of our AMICO middleware, the functions defined in spreadsheets can easily be adapted and used with other message-oriented middlewares (e.g., Linda [24] has pretty much the same primitives, i.e., read and write).

What makes a difference is the type of coordination entities supported by the middleware. The existing middlewares and message-oriented systems usually require uniformity of communication interfaces, which limits the number of services that can be used but enables easy integration of the services that already support such interface.

We are also exploring how AMICO and similar message-oriented service middlewares could be used with other end-user environments, such as graphical diagram editors used in environments such as Matlab or EyesWeb [10]. For example, we used the EyesWeb graphical editor to define image motion processing, where the system is tracking the center of mass of the animated human body. Using existing network sender and receiver primitives of this environment, we connected the example to the AMICO middleware, so that it can receive the URL to the video that has to be processed from other AMICO modules and send the coordinates of the center of the detected figure to AMICO.

We are also exploring how spreadsheets can be used in combination with higher level scripting languages. Scripting languages are usually easier to learn by end-user developers as they use typeless approaches to achieve a higher level of programming enabling more rapid application development than system programming languages. Our middleware currently also supports several higher level scripting languages, including Javascript, Python, BeanShell, Groovy, Ruby, TCL, Sleep, Haskell, and Prolog. Our implementation of scripting support is based on the Java Scripting Project, except for the support for Prolog, which is based on the JLog project.

8.6 Relation with Conventional Service Composition Solutions

In the end, it is important to note that our solution is aimed not as a replacement but as a complement to existing conventional service composition solutions such as those described in Section 3.2. We complement these solutions in two aspects:

- enabling nondevelopers to exploit software services without programming and the need to learn complex protocols and data structures and
- providing a simple environment where the functionality of services can be explored and various service combinations could be tested in the early stages of development.

With AMICO:CALC, end users can directly exploit many available software services without the need to learn a new language or change their EUD environment. On the other hand, exploration of the space of service combinations is important in domains such as the design of interactive systems, where it may not be clear in the early stages of development what services are necessary and how the user will accept them. Building prototypes with conventional service composition approaches may be expensive and time consuming. By reducing development efforts and by making it possible to involve end users in the early stages of development, our solution can facilitate the exploration of the space of possible service compositions before the decision about the final implementation is made.

9 CONCLUSION

We have presented a framework for a spreadsheet-based composition of heterogeneous software services. The main contribution of our work is showing how spreadsheets, a paradigm proven to be highly productive and simple to learn and use, can be used for complex service compositions. We have described the details of our framework for spreadsheet-based service composition and discussed the identified end-user issues based on its use in several projects and in education.

Providing services directly to end users may also promote more frequent use of services, possibly changing the business model from business to business (B2B or developer to developer) to business to consumer (B2C). With millions of end-user developers, the number of potential service consumers is much higher. This shift is also characteristic in other service industries, opening new opportunities for the service economy [48].

In the future work, we will continue to address many of the open issues such as debugging, adding tools for discovery of services, automatic integration of services without manual configuring of adapters, and security. As we have integrated our solutions into existing mainstream EUD environments, we also plan to reuse other solutions that work with these environments, such as debuggers [2], [46], or work with assertions [8]. An interesting research direction is to explore integrations with other EUD environments, including visual programming tools. We also plan to compare our (end-user-oriented) approach with other (more developer-oriented) service composition and business process modeling approaches (e.g., BPEL).

ACKNOWLEDGMENTS

This research was partially supported by the European Commission under contract FP6-027026, project-K-Space. The authors thank Anton Eliens and Farhad Arbab for

providing useful feedback on the work described here and for comments that significantly improved this article.

REFERENCES

- [1] R. Abraham and M. Erwig, "Inferring Templates from Spreadsheets," *Proc. 28th Int'l Conf. Software Eng. (ICSE '06)*, pp. 182-191, 2006.
- [2] R. Abraham and M. Erwig, "GoalDebug: A Spreadsheet Debugger for End Users," *Proc. 29th Int'l Conf. Software Eng. (ICSE '07)*, pp. 251-260, 2007.
- [3] A. Ambler, "The Formulate Visual Programming Language," *Dr. Dobbs J.*, vol. 24, no. 8, pp. 21-28, 1999.
- [4] B. Buxton, *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann, Mar. 2007.
- [5] B. Boehm et al., "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0," *Annals of Software Eng.*, special volume on software process and product measurement, pp. 57-94, 1995.
- [6] B.A. Nardi, *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, July 1993.
- [7] P.S. Brown and J.D. Gould, "An Experimental Study of People Creating Spreadsheets," *ACM Trans. Information Systems*, vol. 5, no. 3, pp. 258-272, 1987.
- [8] M. Burnett et al., "End-User Software Engineering with Assertions in the Spreadsheet Paradigm," *Proc. 25th Int'l Conf. Software Eng. (ICSE '03)*, pp. 93-103, 2003.
- [9] M.M. Burnett and H.J. Gottfried, "Graphical Definitions: Expanding Spreadsheet Languages through Direct Manipulation and Gestures," *ACM Trans. Computer-Human Interaction*, vol. 5, no. 1, pp. 1-33, 1998.
- [10] A. Camurri, M. Ricchetti, and R. Trocca, "EyesWeb—Toward Gesture and Affect Recognition in Dance/Music Interactive Systems," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems (ICMCS '99)*, vol. 1, pp. 643-648, <http://www.eyesweb.org/>, 1999.
- [11] S.-K. Chang, *Principles on Visual Programming Systems*. Prentice Hall, 1990.
- [12] E.H.H. Chi, J. Riedl, P. Barry, and J. Konstan, "Principles for Information Visualization Spreadsheets," *IEEE Trans. Computer Graphics and Applications*, vol. 18, no. 4, pp. 30-38, 1998.
- [13] P. Ciancarini, "Coordination Models and Languages as Software Integrators," *ACM Computing Surveys*, vol. 28, no. 2, pp. 300-302, 1996.
- [14] F. Curbera et al., "Unraveling the Web Services Web," *IEEE Internet Computing*, vol. 6, no. 2, pp. 86-93, 2002.
- [15] *EditGrid Online Spreadsheets Web Page*, <http://www.editgrid.com/>, 2008.
- [16] *End Resources on End-User Software Engineering*, 2008.
- [17] *Comm. ACM*, special issue on end-user development, vol. 47, no. 9, pp. 31-94, Sept. 2004.
- [18] E. Freeman, S. Hupfer, and K. Arnold, *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley Professional, June 1999, ISBN 0-201-30955-6.
- [19] F.J. Lerch, M.M. Mantei, and J.R. Olson, "Skilled Financial Planning: The Cost of Translating Ideas into Action," *Proc. ACM SIGCHI Conf. Human Factors in Computing Systems: Wings for the Mind (CHI '89)*, pp. 121-126, 1989.
- [20] G. Fitzpatrick et al., "Augmenting the Workaday World with Elvin," *Proc. Sixth European Conf. Computer Supported Cooperative Work (ECSCW '99)*, pp. 431-450, 1999.
- [21] *Flickr Web Services Site*, <http://www.flickr.com/services/api/>, 2008.
- [22] J. Fujima, A. Lunzer, K. Hornbæk, and Y. Tanaka, "Clip, Connect, Clone: Combining Application Elements to Build Custom Interfaces for Information Access," *Proc. 17th Ann. ACM Symp. User Interface Software and Technology (UIST '04)*, pp. 175-184, 2004.
- [23] E. Gamma, R. Helm, R. Johnson, and J. Vlisside, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Nov. 1994.
- [24] D. Gelernter, "Generative Communication in Linda," *ACM Trans. Programming Languages and Systems*, vol. 7, no. 1, pp. 80-112, Jan. 1985.
- [25] *Google Web Spreadsheets Web Site*, <http://docs.google.com/>, 2008.
- [26] S. Greenberg and B. Buxton, "Usability Evaluation Considered Harmful (Some of the Time)," *Proc. 26th Ann. ACM SIGCHI Conf. Human Factors in Computing Systems (CHI '08)*, pp. 111-120, 2008.
- [27] *HandVu Hand Gesture Recognizer Project Web Site*, <http://www.movesinstitute.org/~kolsch/HandVu/HandVu.html>, 2008.
- [28] M.N. Huhns and M.P. Singh, "Service-Oriented Computing: Key Concepts and Principles," *IEEE Internet Computing*, vol. 9, no. 1, pp. 75-81, 2005.
- [29] *iRows Online Spreadsheets Web Page*, <http://www.irows.com/>, 2008.
- [30] B. Johanson, A. Fox, and T. Winograd, "The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms," *IEEE Pervasive Computing*, vol. 1, no. 2, pp. 67-74, Apr. 2002.
- [31] E. Kandogan, E. Haber, R. Barrett, A. Cypher, P. Maglio, and H. Zhao, "A1: End-User Programming for Web-Based System Administration," *Proc. 18th Ann. ACM Symp. User Interface Software and Technology (UIST '05)*, pp. 211-220, 2005.
- [32] S.S. Laurent, *Microsoft Excel's Web Services*, <http://www.oreillynet.com/pub/h/1306>, 2008.
- [33] J.L. Leopold and A.L. Ambler, "Keyboardless Visual Programming Using Voice, Handwriting, and Gesture," *Proc. IEEE Symp. Visual Languages (VL '97)*, p. 28, 1997.
- [34] E.M. Maximilien and M.P. Singh, "Toward Autonomic Web Services Trust and Selection," *Proc. Second Int'l Conf. Service-Oriented Computing (ICSOC '04)*, pp. 212-221, 2004.
- [35] B.A. Myers, "Graphical Techniques in a Spreadsheet for Specifying User Interfaces," *Proc. ACM SIGCHI Conf. Human Factors in Computing Systems (CHI '91)*, pp. 243-249, 1991.
- [36] *NumSum Online Spreadsheets Web Page*, <http://numsum.com/>, 2008.
- [37] Ž. Obrenovic and D. Gašević, "Open-Source Software: All You Do Is Put It Together," *IEEE Software*, vol. 24, no. 5, pp. 86-95, 2007.
- [38] Ž. Obrenovic, D. Gašević, and A. Eliëns, "Stimulating Creativity through Opportunistic Software Development," *IEEE Software*, vol. 25, no. 6, pp. 64-70, Nov./Dec. 2008.
- [39] *OpenCV Computer Vision Library Project Web Site*, <http://opencvlibrary.sourceforge.net/>, 2008.
- [40] *OpenID Web Site*, <http://openid.net/>, 2008.
- [41] B. Orriëns, J. Yang, and M.P. Papazoglou, "Model Driven Service Composition," *Proc. First Int'l Conf. Service-Oriented Computing (ICSOC)*, 2003.
- [42] G.A. Papadopoulos and F. Arbab, "Coordination Models and Languages," *Advances in Computing—The Eng. of Large Systems*, M. Żelkowitz, ed., vol. 46, Academic Press, 1998.
- [43] M.P. Papazoglou and D. Georgakopoulos, "Service-Oriented Computing," *Comm. ACM*, vol. 46, no. 10, pp. 25-28, 2003.
- [44] J.F. Patterson, M. Day, and J. Kucan, "Notification Servers for Synchronous Groupware," *Proc. ACM Conf. Computer Supported Cooperative Work (CSCW '96)*, pp. 122-129, 1996.
- [45] J. Rao, P. Küngas, and M. Matskin, "Composition of Semantic Web Services Using Linear Logic Theorem Proving," *Information Systems*, vol. 31, nos. 4-5, p. 34, June/July 2006.
- [46] J.R. Ruthruff, M. Burnett, and G. Rothermel, "An Empirical Study of Fault Localization for End-User Programmers," *Proc. 27th Int'l Conf. Software Eng. (ICSE '05)*, pp. 352-361, 2005.
- [47] C. Scaffidi, M. Shaw, and B. Myers, "Estimating the Numbers of End Users and End User Programmers," *Proc. IEEE Symp. Visual Languages and Human-Centric Computing (VLHCC '05)*, pp. 207-214, 2005.
- [48] *Comm. ACM*, special issue on service sciences, vol. 49, no. 7, pp. 30-87, July 2006.
- [49] N.C. Shu, "Visual Programming: Perspectives and Approaches," *IBM Systems J.*, vol. 28, pp. 525-547, 1989.
- [50] *Simple Spreadsheets Platform Home Page*, <http://www.simplegroupware.de/cms/Spreadsheet/Home>, 2008.
- [51] *StrikeIron SOA Express for Excel Web Page*, http://www.strikeiron.com/tools/tools_soexpress.aspx, 2008.
- [52] *Skipper Project Web Site*, <http://www.sxip.com/skipper>, 2008.
- [53] S.H.T. Thompson and M. Tan, "Quantitative and Qualitative Errors in Spreadsheet Development," *Proc. 30th Hawaii Int'l Conf. System Sciences (HICSS '97)*, p. 149, 1997.
- [54] W.K. Edwards, "Putting Computing in Context: An Infrastructure to Support Extensible Context-Enhanced Collaborative Applications," *ACM Trans. Computer-Human Interaction*, vol. 12, no. 4, pp. 446-474, 2005.
- [55] G. Wang and A. Ambler, "Solving Display-Based Problems," *Proc. IEEE Symp. Visual Languages (VL '96)*, p. 122, 1996.
- [56] *wikiCalc Home Page*, <http://www.softwaregarden.com/products/wikicalc>, 2008.

- [57] N. Wilde and C. Lewis, "Spreadsheet-Based Interactive Graphics: From Prototype to Tool," *Proc. ACM SIGCHI Conf. Human Factors in Computing Systems (CHI '90)*, pp. 153-160, 1990.
- [58] *Workflow Patterns Web Portal*, <http://www.workflowpatterns.com/>, 2008.
- [59] J. Yang, B. Orriens, and M.P. Papazoglou, "A Framework for Business Rule Driven Service Composition," *Proc. Fourth Int'l Workshop Conceptual Modeling Approaches for E-business Dealing with Business Volatility*, 2003.
- [60] Y. Le Blevec, C. Ghedira, D. Benslimane, and X. Delatte, "Service-Oriented Computing: Bringing Business Systems to the Web," *IT Professional*, vol. 9, no. 3, pp. 19-24, 2007.
- [61] L.J. Zhang, S. Ericksen, and J. Roy, "A Web 2.0 Tune-Up," *IT Professional*, vol. 9, no. 3, p. 9, May/June 2007.
- [62] *Zoho Online Spreadsheets Web Page*, <http://sheet.zoho.com/>, 2008.



Željko Obrenović received the PhD degree in computer science from the University of Belgrade. He is an assistant professor with the User Centered Engineering Group, Department of Industrial Design, Technische Universiteit Eindhoven (TU/e), The Netherlands. His research interests include human-computer interaction and software engineering. The work presented in the paper was realized during his work at CWI, Amsterdam.



Dragan Gašević received the BS, MSc, and PhD degrees in computer science from the University of Belgrade. He is an assistant professor and ingenuity new faculty in the School of Computing and Information Systems, Athabasca University, Athabasca, Alberta, Canada. His research interests include the Semantic Web, model-driven software engineering, knowledge management, service-oriented architectures, and learning technologies.