Open Source Software: All You Do Is Put It Together

Željko Obrenović, *CWI Amsterdam*

Dragan Gašević, *Athabasca University*

An infrastructure for rapid prototyping with open source software components focuses on pragmatic aspects of OSS integration. Two examples demonstrate the infrastructure's use in complex scenarios.

As of 1 August 2007, SourceForge.net hosted more than 150,000 registered open source software projects, and many more projects are available on other sites. With so many OSS choices, it might seem that building a new application is only a matter of finding the appropriate projects and putting them together. In fact, this idea inspired our subtitle, borrowed from a famous 1938 Walt Disney cartoon. In "Boat Builders," Mickey Mouse, Donald Duck, and Goofy try to construct a ship, "Queen Minnie," from folding parts. They have a blueprint and all the necessary components. So, according to Mickey, "All you do is put it together." But that proves easier said than done. Some parts behave in unexpected ways, and others simply won't fit together. In the end, when the ship is launched, everything begins to collapse, and once out at sea, it separates into its parts again.

The experience of Mickey, Donald, and Goofy resembles that of many developers who build software using open source components. OSS projects cover a wide range of topics but often use incompatible technologies. Furthermore, even though the topics might address all developers' needs, new features and other modifications primarily address project contributors' interests and wishes.[1] The contributors' needs and experience will also drive the choice of development environment and platform.[2] If you want to use OSS widely, you and your team must master several programming languages, interfaces, or network protocols. In other words, even though you get OSS components for free, integrating them usually requires significant time and effort.[3] This is especially problematic in early development phases when it's often unclear whether a system is feasible or acceptable to potential users, and you want to rapidly prototype many alternatives quickly and with minimal effort.[4]

We've developed the Adaptable Multi-Interface Communicator infrastructure to support rapid prototyping from OSS components. AMICO is based on existing middleware platforms for component integration, but it focuses on pragmatic aspects of OSS integration—often absent from existing integration platforms. AMICO satisfies requirements based on our experiences in solving practical problems in several projects.

Middleware infrastructure requirements

Introducing a flexible middleware infrastructure can facilitate component integration and significantly speed up application deploy-

0740-7459/07/$25.00 © 2007 IEEE

ment.[5] However, existing integration middleware often poses problems for OSS components. Although T.R. Madanmohan and Rahul De' argued that OSS gives organizations new options for component-based development similar to commercial off-the-shelf software,[1] using existing COTS middleware systems to support OSS component integration is very difficult. OSS components use such diverse technologies, protocols, and implementation platforms that wrapping the code in COTS-compliant packages is practically impossible.[5] (The "Ambient Intelligence in Interactive TV" sidebar illustrates this diversity in relation to the AMICO examples we describe later.)

Loosely coupled integration seems more appropriate for OSS components because it addresses this heterogeneity and enforces fewer restrictions on the integration. Many application areas have developed infrastructures for loosely coupled component integration. These infrastructures usually provide a shared data store with notification services. For example, the Elvin system[6] is a well-known pure notification service (without data storage), applied successfully to many collaborative applications. The Lotus PlaceHolder system[7] is another notification service, which additionally incorporates a persistent data store that applications can use to manage shared data. Any client changes to this data generate notifications to other interested clients.

Several data repositories have models similar to these notification services. For example, tuplespace systems such as Linda,[8] the Stanford EventHeap,[9] and JavaSpaces[10] let applications store untyped tuples of named data elements in the model. However, most of these systems focus on collaborative applications or context-aware computing and don't easily adapt to other domains.

Infrastructures for service-oriented and related computing applications take a similar approach. For example, the Service-Oriented Device Architecture models devices as services and embeds them on an enterprise service bus.[11] In this way, SODA makes device access and control available to a wide range of enterprise applications. Service-oriented architectures are, however, based on unified interfaces and relatively complex integration standards.

Most existing integration middleware supports only a limited number of programming interfaces, while OSS components use a diverse set of programming languages and communication protocols.

Our experience shows that rapid application development with a broader set of OSS components requires a middleware integration infrastructure that offers the following:

Ambient Intelligence in Interactive TV

As part of the Information Technology for European Advancement (ITEA) Passepartout project (www.passepartout-project.org), we've been exploring novel user interfaces in the interactive TV domain. One project goal was to explore available components for building ambient-intelligence solutions and to develop recommendations for their use. Because predicting how users might accept a particular solution was difficult, we had to rapidly prototype many systems and evaluate them to see how well they supported the project's vision.

The OSS community has developed many components that support novel interaction modalities. Computer vision alone includes hundreds of freely available pieces of software (see www.cs.cmu.edu/~cil/v-source.html). We've used several of these components, such as the Open Computer Vision Library (http://opencvlibrary.sourceforge.net), a collection of algorithms and sample code for computer-vision problems. The HandVu project (www.movesinstitute.org/~kolsch/HandVu/HandVu.html) supports a vision-based hand-gesture interface. There are also several open source speech-recognition platforms, such as CMU Sphinx-4 (http://cmusphinx.sourceforge.net/sphinx4), as well as many text-to-speech platforms—for example, FreeTTS Text-to-Speech Synthesizer (http://freetts.sourceforge.net) and NeXTeNS Text-to-Speech Synthesizer for Dutch (http://nextens.uvt.nl).

Among the many open source AI-based interface solutions, Sesame (www.openrdf.org) is a Resource Description Framework database, supporting RDF Schema inferencing and querying. KAON (http://kaon.semanticweb.org) is an infrastructure for managing Web Ontology Language-Description Logics, Semantic Web Rule Language, and F-Logic ontologies. ConceptNet (http://web.media.mit.edu/~hugo/conceptnet) is a common-sense knowledge base and natural-language-processing toolkit that supports many practical, real-world, textual-reasoning tasks.

We also used many open source interactive multimedia components. VLC (www.videolan.org) is a highly portable multimedia player for various audio and video formats. Ambulant Player (www.cwi.nl/projects/Ambulant) is a media player that supports the Synchronized Multimedia Interchange Language. VeeJay (http://veejay.dyne.org) is a visual instrument and video sampler that lets users play and mix the video in real time.

These components use various programming languages, such as Java, C++, and Python. Many of the projects offer interfaces that can make their integration easier. For example, ConceptNet, LifeNet, and the Ambulant Player run XML-RPC servers, enabling other servers to access their functionality without linking their code. Multimedia tools, such as VeeJay, offer the Open Sound Control interface. NeXTeNS and HandVu offer TCP interfaces. You can control the VLC player through a built-in HTTP or Telnet server. Other components don't provide explicit service interfaces, so you must use them as libraries to compile and link with your applications.

The Amico infrastructure provides a unified view of different communication interfaces.

- *Support multiple integration interfaces and enable integration of new ones.* Components from open source projects use diverse integration interfaces, none of which is predominant. Adapting components to one common interface is never an easy task, and it's sometimes impossible.
- *Enable flexible integration.* Existing component integration systems often require developers to agree on many rules, such as naming conventions. Infrastructures that require such agreements restrict component reuse.
- *Bridge data and temporal gaps.* Low-level components, such as sensors, and higher level components, such as Web services, work with significantly different data structures and temporal constraints. For example, face-detector sensors can send dozens of UDP packets per second with simple data structures about detected events, while Web services use the more complex HTTP transport and XML-encoded data, incurring delays that sometimes measure in seconds. To integrate such components, the infrastructure must be able to abstract and map different data types supporting temporal functions, such as frequency filtering.
- *Support fault tolerance.* Many OSS components are still under development and often unstable. The infrastructure must therefore ensure that components that crash won't cause other components to crash.
- *Use open standards.* Standardization is a significant driving force for progress because it codifies best practices, enables and encourages reuse, and facilitates interworking between complementary tools.
- *Reduce the initial effort to develop a simple component.* In other words, make simple things simple. Most COTS middleware infrastructures are powerful, but the initial effort to develop a simple component can be high.

The Amico infrastructure

Amico is a generic infrastructure that meets these requirements so that developers can rapidly prototype and experiment with OSS components. It's available as an open source project at http://amico.sourceforge.net, where you'll also find many examples.

Basic concepts

Amico OSS is based on a publish-subscribe infrastructure for integrating loosely coupled services. In such infrastructures, a publisher updates a shared data repository. The publisher might or might not know or care whether any subscribers are listening for updates. The loosely coupled approach can be highly adaptable when using simple data structures, because new applications can use existing data in the model and add their own without breaking the infrastructure. Components communicate by exchanging events through a shared data repository consisting of named slots called *variables*. Components can update the variables and register for notifications about variable changes. Modules can also derive new variables by processing existing ones.

Figure 1 shows a UML class diagram of basic concepts of our integration infrastructure. Its core is the shared data repository called the communicator. Communication adapters update basic variables and register templates. The communicator registers each template with variables that trigger its evaluation and with a string that can replace parts with variable values. For example, a template string `"SELECT <%=fields%> FROM <%=table%>"` will return an SQL query that replaces `<%=fields%>` and `<%=table%>` with actual values of variables named `fields` and `table`. Each adapter receives a notification with populated templates (that is, templates whose variables are replaced with actual values). This notification will cause some adapter-specific action—for example, invoking the procedure whose parameter is the `SQL SELECT` statement.

Integration support for multiple interfaces

Up to this point, the infrastructure's core resembles regular notification services. A first key difference derives from our requirement to support multiple integration interfaces. The Amico infrastructure provides a unified view of different communication interfaces, implementing a common space for interconnecting them. As figure 2 shows, we've supported several widely used communication protocols, such as XML-RPC, Open Sound Control (OSC), URL, and SOAP, as well as many application-specific adapters (see figure 2a). The infrastructure is extensible, so developers can add new communication interfaces. Figure 2b and 2c show Java code fragments for updating infrastructure variables by using XML-RPC and OSC interfaces, respectively. Figure 2d

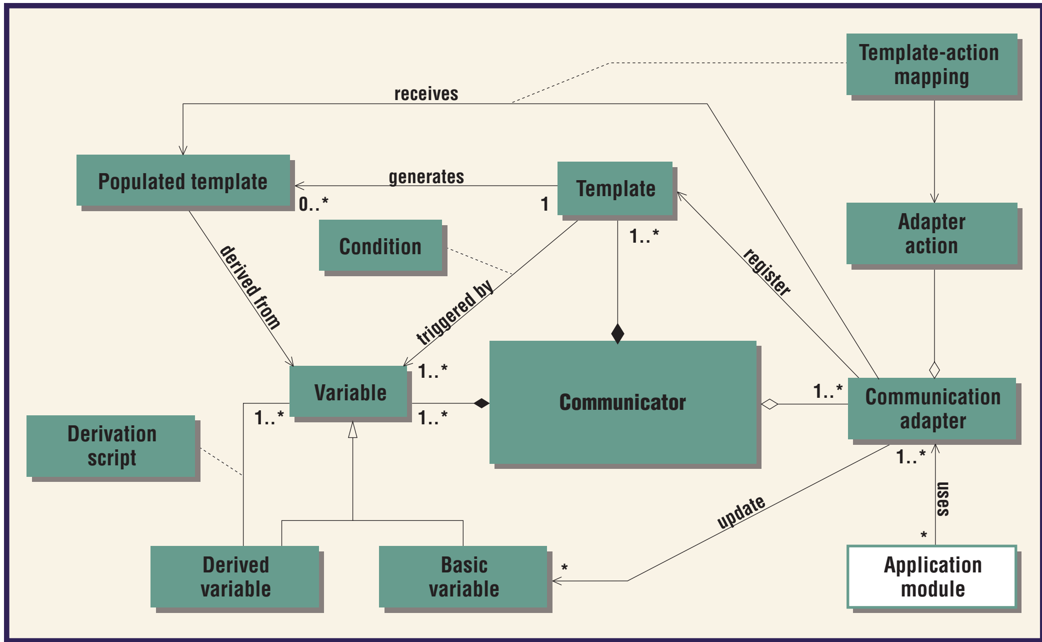

Figure 1. A class diagram of Amico integration infrastructure concepts.

and 2e show fragments of the adapter's configuration files for mapping variable updates to XML-RPC method calls and OSC messages, respectively.

One main reason Amico supports multiple interfaces is that many developers are familiar with only one interface standard and, sometimes, haven't even heard of others. In these cases, it was easier to extend Amico with a new integration interface than to adapt the components to one common interface.

Flexible component integration

Amico enables flexible system configuration through the derivation of new named slots, or *variables*. New variables are derived using a set of *transformations*, defined in XSLT, a standardized, commonly used transformation script language.[12] Figure 3 presents an example XSLT fragment transformation that derives the variable `distance-rank` from the variable `distance`.

An important goal of our transformation framework is to enable variable transformations in several steps. For example, in sensing-based systems, you can start from low-level sensor data and transform it into intermediate variables. Other transformations can then use these intermediate variables to derive application-specific variables. Using intermediate variables can also simplify integration of new sensing modules: you can reuse transformations from intermediate to application-specific variables and add only transformations of application-specific variables to intermediate variables. Although components can use the same variables, we propose deriving variables in several layers in accordance with the ideas of model-driven transformations.[13]

The infrastructure also lets you dynamically load and unload transformation scripts. In this way, you can reconfigure the infrastructure on the fly, which lets you support, for example, interaction with a newly registered device or a user's interaction modality preferences.

Bridging semantic and temporal gaps

The transformations also provide the means for bridging the different abstraction levels found in different data structures. Transformations can abstract low-level data so that an application doesn't receive all low-level events. For example, it might receive notification only when a user enters a certain area in a camera's visual field or exceeds a motion-detection threshold.

Transformations can also solve the problem of different component temporal constraints. Transformations can use time stamps to derive new slots that are updated with lower frequency, allowing higher level modules to ignore other updates. This can simplify modules because they don't have to change to meet other modules' synthetic and temporal constraints. It also makes module reuse easier.

Because Amico can work with significantly

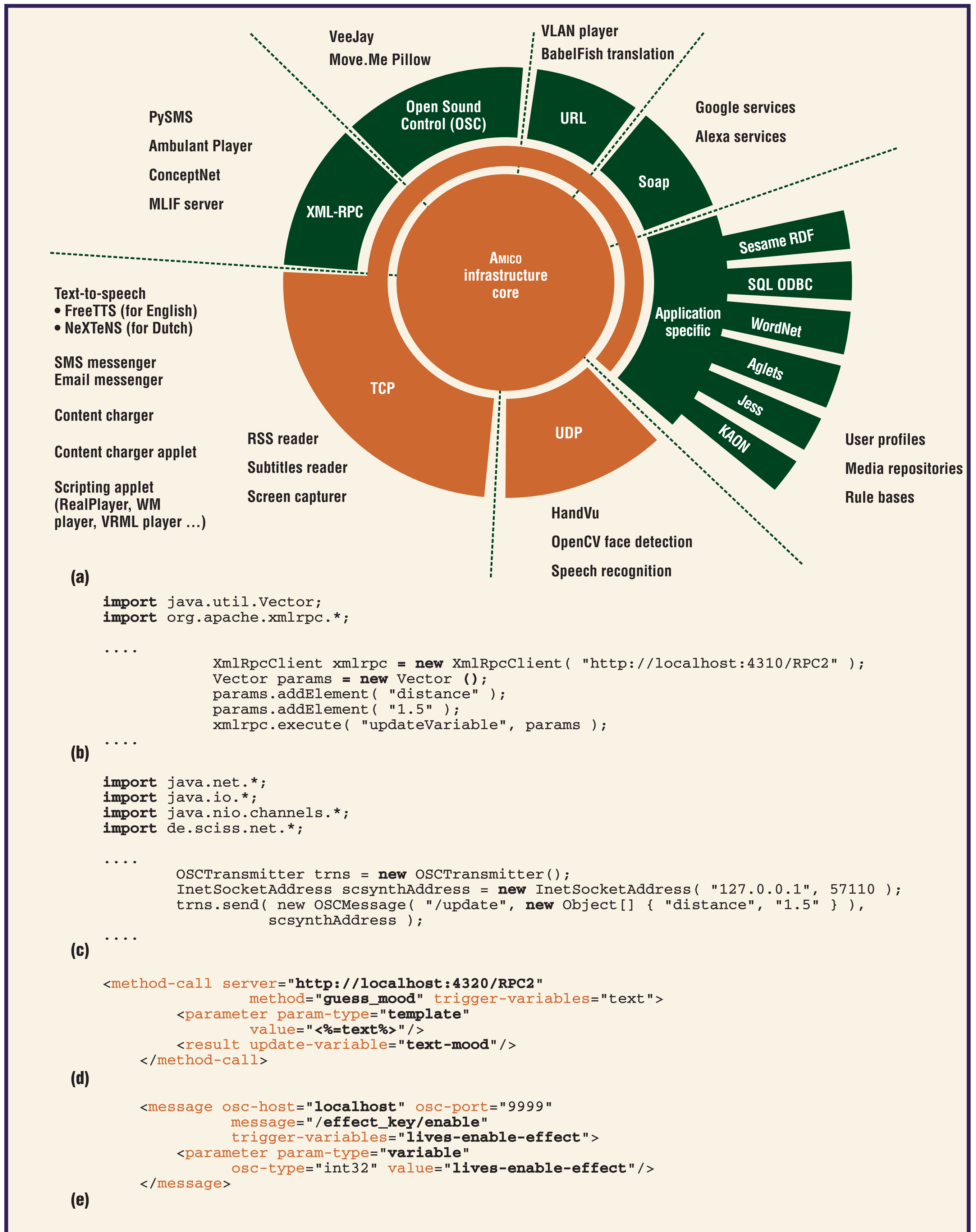

Figure 2. Example AMICO interfaces and code fragments. (a) The infrastructure supports a wide range of communication interfaces (orange) and many application-specific adapters (green). Java code fragments show how to update variables using (b) XML-RPC and (c) OSC interfaces as well as adapter configuration files for mapping variables updates to (d) XML-RPC method calls and (e) OSC messages.

different data and temporal constraints, we were also able to use it as a simple integrator for existing Web services. For example, we integrated Google and Yahoo search services with other (lower level) components.

Other requirements

AMICO's loosely coupled approach is inherently fault tolerant. Components run as independent processes. So, if one component crashes, it doesn't affect other components, even though the functionality of the system as a whole is usually affected.

The basic AMICO technologies include XML, Java, and Internet protocols, all using open standards. Using open standards also enables reuse of existing solutions, such as XML/XSLT editors.

The infrastructure requires neither significant changes to existing OSS components nor specialized and complex tools to tailor new applications. AMICO provides a declarative XML platform abstraction, configuring all communication interfaces and transformations in XML/XSLT files. This enables system configuration with ordinary text editors and without compilation. We've also created tools to help both developers and end users. For example, AMICO:CALC is an OpenOffice Calc extension that lets users rapidly configure and connect components through a spreadsheet.

```
<variables>
    <variable name="distance">
          1.5
    </variable>
</variables>
```
(a)

```
<xsl:template match="/variables/variable[@name='distance']">
    <variable name='distance-rank'>
        <xsl:if test="current() >= 1">ok</xsl:if>
        <xsl:if test="current()  < 1">too-close</xsl:if>
    </variable>
</xsl:template>
```
(b)

```
<variables>
    <variable name="distance">
          1.5
    </variable>
    <variable name="distance-rank">
          ok
    </variable>
</variables>
```
(c)

Figure 3. An example XSLT fragment: (a) basic variable `distance`, (b) variable transformation to `distance-rank`, and (c) derived and basic variables.

Two example implementations

With the AMICO infrastructure, we've integrated dozens of OSS components and rapidly prototyped many interactive solutions. Two relatively complex examples illustrate the use of diverse components through diverse interfaces.

Figure 4a shows an application configuration that uses camera-based face detection to control the playback of multimedia player components. This example also uses an RFID reader, the Sesame RDF server, a multilingual server (MLIF) developed by one of our partners, several text-to-speech (TTS) engines, as well as several multimedia players. The RFID reader is a hardware component, but the AMICO framework lets us treat it as an RS-232 (serial) interface component. We developed a simple adapter to map the RS-232 communication protocol to TCP. This lets the RFID reader update a variable through detected user IDs. The application then uses these IDs to derive a query for the Sesame RDF repository, which contains user profiles. The Sesame adapter updates a new variable with an XML-encoded user profile. Using a built-in XML processor, the application then extracts more atomic values from user profiles, such as user language or age. The language profile determines appropriate messages and TTS engines.

The face detector determines how many people are in front of the screen and their distances from it. The application uses these parameters to control the playback of multimedia players and to send messages to the user. If no one is in front of the screen, the playback stops. It starts up again and continues if the face detector senses at least one person. If a person is too close to the screen, then playback stops again, and the application speaks to the user through a TTS engine. The MLIF prepares the message in the user's language. The application runs several TTS engines simultaneously. Each engine is registered for different application-specific variables derived from the language-specific message. Each time, the application derives just one of these variables, triggering just the TTS engine that can support the user's language. If no TTS engine exists for a specific user language, the application defaults to the English engine.

Figure 4b illustrates the integration of more complex devices. Institute V2_ from Rotterdam is developing a biometric pillow with em-

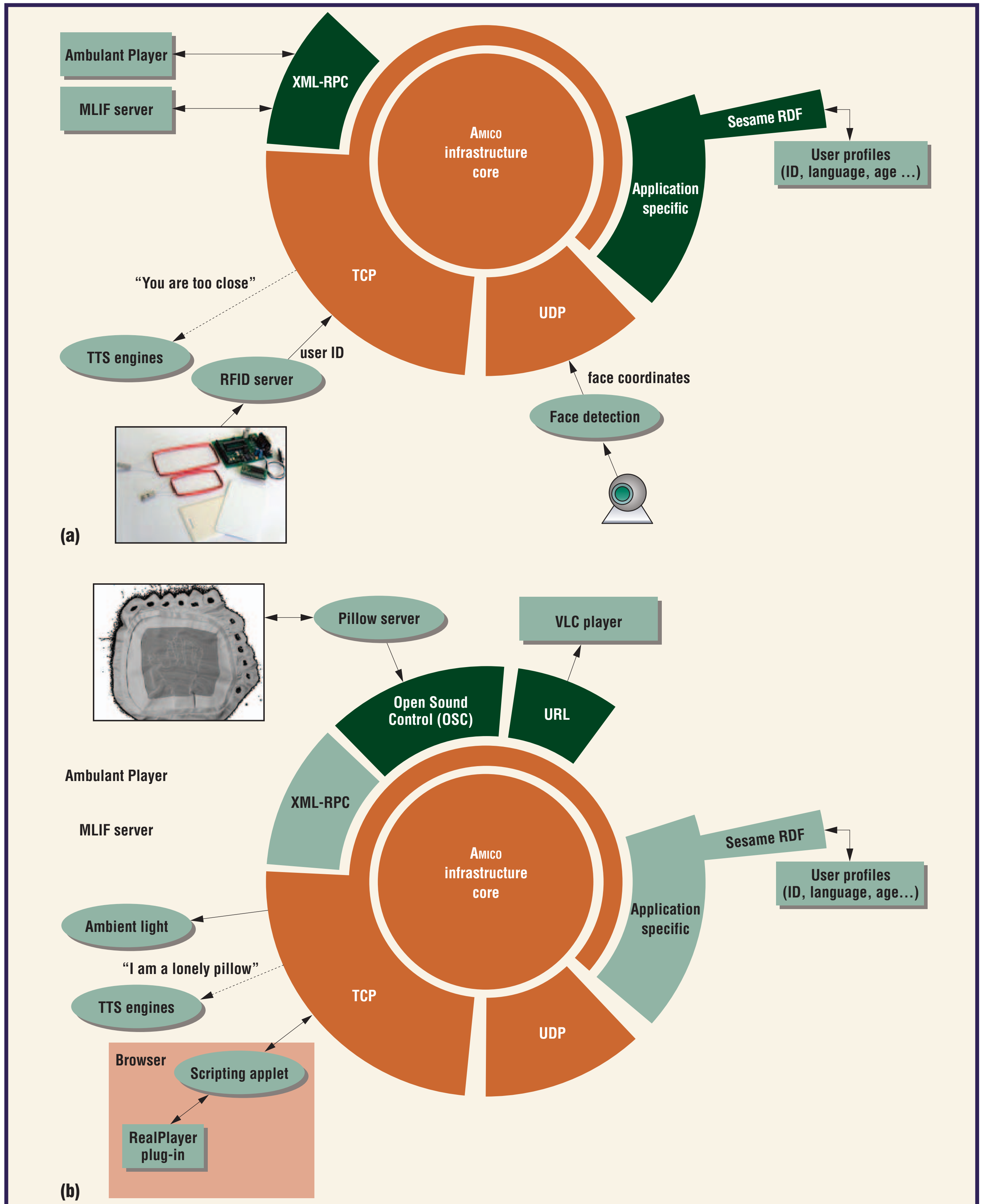

Figure 4. Two example configurations of the Amico infrastructure: (a) camera-based face detection used to control the playback of the multimedia players and (b) integration of more complex devices.

bedded sensors for pressure, galvanic skin response, movement patterns, and presence. We integrated this system with the Amico infrastructure through the OSC interface. The system also uses this interface for communication between the pillow's hardware and driver software. In this scenario, we configured the system so that pressure on different pillow areas generates discrete actions. The system then maps these actions to multimedia player commands that control sound intensity and playback and send a message to the user. We reused the components from the first example by adding a mapping from one variable—a variable that the pillow updates through the OSC interface—to other variables.

All the components used in these examples are relatively simple and unaware of other components. They update the infrastructure variables through the interfaces that are easiest to implement. For each scenario, we've provided different configurations with variable transformations. For example, figure 5 illustrates variable transformations from our first example. The face detector updates its application-specific variable by sending coordinates of detected faces over a UDP interface. The infrastructure then runs transformations that derive intermediate variables: one describing the number of detected faces and, for each face, a variable containing its coordinates. The number of faces directly correlates to platform-independent variables representing the number of people in the room. The face coordinates facilitate derivation of the average face height, which roughly correlates to the distance of people from the camera. The system can then use the derived distance variable to change the variables used to control content presentation, such as volume intensity or font size.

Although we could directly obtain playback controls from a sensor variable, deriving variables in several layers is more flexible. For example, we could have derived the playback control from a speech recognizer that updates the same variable. In this way, we could control the player through speech or motion, but we wouldn't have to reconfigure the system because all the sensing modules update the same variables. They just use different transformations. Depending on the media player used, the system can transform playback control variables into XML-RPC function calls (Ambulant Player), HTTP GET requests (VLC player), OSC messages (VeeJay), or TCP messages (Scripting applet).

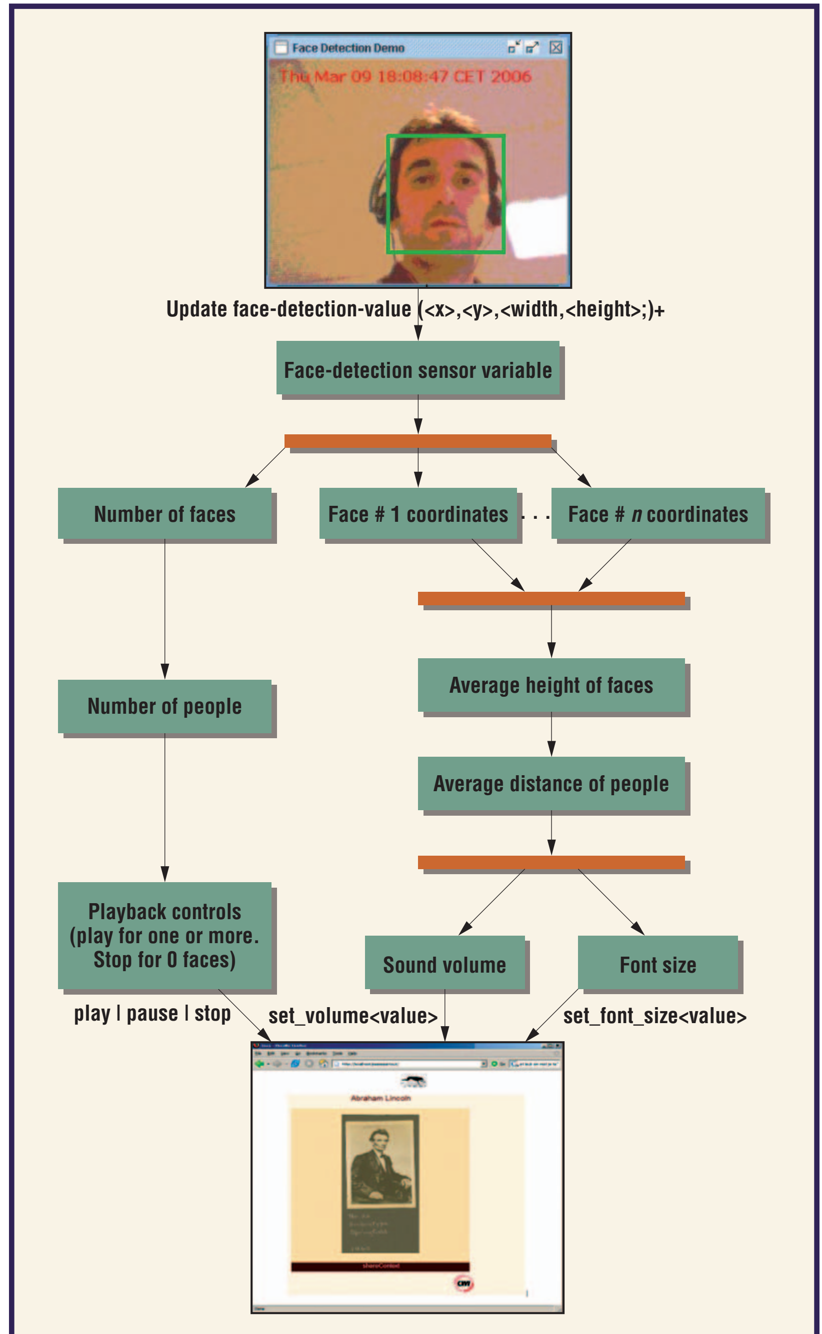

Figure 5. Derivation of variables in a camera-based multimedia playback control. The face detector updates its application-specific variable by sending the coordinates of faces it detects. The infrastructure then runs transformations that derive several other intermediate variables.

Discussion

Our work raises two questions: one regarding its implications for OSS developers who

Including standalone examples with your distributions makes rapid prototyping easier.

wish to make their software easier to use with other applications and the other regarding observed performance issues.

Simplifying OSS component integration

When OSS components offer their functionality through open communication interfaces, integration with AMICO is straightforward: you just edit configuration files. When the components don't do so, you must implement adapters to turn the component into a simple standalone service. To make rapid prototyping easier, we encourage OSS developers to include such standalone service-oriented examples with their distributions. Building, installing, and running standalone programs is usually straightforward, even for users who aren't familiar with the technology that a component uses—that is, even if you don't know Python or Java, it's still relatively easy to install Python or Java interpreters and to build and run Python or Java applications). When you include service-oriented examples with your distributions, applications written in other languages can use your components, even without middleware infrastructures such as AMICO. For example, you can expose a component's functions with the Python OSC server library, which other components can access through Java or C++ OSC client libraries.

Adapting OSS components to standalone services doesn't require changing the basic component functionality. You need only add code that offers the functionality through any of the supported open communication interfaces, such as those shown in figure 2. As a starting point for these adaptations, we've often used examples and demo programs that come with OSS distributions. For instance, OpenCV comes with several demo programs, illustrating many of the library's possible uses. We've adapted several of these demo programs by adding a simple part to connect the component with AMICO. For example, in the face-detection example, an original demo program detected the faces and wrote the coordinates on the console. We modified this program into a simple "push" server that, instead of writing data to the console, sent the detected coordinates as UDP packages to AMICO using a C socket library. For components that require bidirectional and more complex communication, we usually use TCP-based interfaces, such as XML-RPC.

AMICO performance issues

The communication interfaces and mediated interaction reduce system performance compared with more tightly coupled components. The exact performance overhead depends on the component network distribution, the AMICO transformation complexity, the integration interfaces, and the OSS component memory and processor overhead. AMICO comes with several tools that can help developers measure delay and identify performance bottlenecks. For the example applications described earlier, performance has been satisfactory, introducing processing overhead of less than 50 microseconds.

Although this delay is acceptable for most interactive applications, AMICO isn't suitable for real-time applications. It makes no guarantees regarding the data transmission speed. Extending its current best-effort model to support performance requirements of real-time applications is a subject of our future work.

The communication interface a developer uses can significantly influence performance. For example, our partners often use XML-RPC because, in many programming languages, it requires adding just a few lines of code. Although much more functional, this interface is significantly slower than the UDP-based interfaces that are often used in real-time applications and games. However, adding such a low-level interface to your application can require significant programming effort if the component exchanges complex data structures.

We're working on connecting AMICO with other environments, such as Web browsers and end-user programming tools, to further support rapid application development. We also plan to use this platform in education, where students can compose complex interactive environments without extensive programming. This project uses an extension of AMICO that supports component integration on the basis of a distributed logic programming paradigm.[14]

We hope our decision to make the AMICO infrastructure an open source project will lead to more applications and encourage further discussions about efficient ways to exploit OSS's huge reusability potential. We encourage OSS developers to make their software easier to integrate by adopting our approach

or something similar. AMICO can cover many developer needs for rapid prototyping and component testing. Moreover, as an open source project, you can adapt it and extend it further. Feel welcome to contribute.

Acknowledgments

The European ITEA Passepartout project and K-Space network of excellence partially funded this research. We thank Bran Selic, Anton Eliëns, and the anonymous reviewers who provided useful feedback on the work described here and whose comments significantly improved this article.

References

1. T.R. Madanmohan and R. De', "Open Source Reuse in Commercial Firms," *IEEE Software*, vol. 21, no. 6, 2004, pp. 62–69.
2. M.J. Karels, "Commercializing Open Source Software," *ACM Queue*, vol. 1, no. 5, July/Aug. 2003, pp. 46–55.
3. M.A. Cusumano, "Reflections on Free and Open Software," *Comm. ACM*, vol. 47, no. 10, 2004, pp. 25–27.
4. Nigel Davies et al., "Rapid Prototyping for Ubiquitous Computing," *IEEE Pervasive Computing*, vol. 4, no. 4, 2005, pp. 15–17.
5. A. Liu and I. Gorton, "Accelerating COTS Middleware Acquisition: The i-Mate Process," *IEEE Software*, vol. 20, no. 2, 2003, pp. 72–79.
6. G. Fitzpatrick et al., "Augmenting the Workaday World with Elvin," *Proc. 6th European Conf. Computer Supported Cooperative Work*, Kluwer Academic Publishers, 1999, pp. 431–450.
7. A.K. Dey et al., "The Conference Assistant: Combining Context-Awareness with Wearable Computing," *Proc. 3rd IEEE Int'l Symp. Wearable Computers*, IEEE CS Press, 1999, pp. 21–28.
8. D. Gelernter, "Generative Communication in Linda," *ACM Trans. Programming Languages and Systems* (TOPLAS), vol. 7 no. 1, Jan. 1985, pp. 80–112.
9. B. Johanson, A. Fox, and T. Winograd, "The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms," *IEEE Pervasive Computing*, vol. 1, no. 2, Apr. 2002, pp. 67–74.
10. Ken Arnold et al., *Jini Specification*, Addison-Wesley, 1999.
11. S. de Deugd et al., "SODA: Service Oriented Device Architecture," *IEEE Pervasive Computing*, vol. 5, no. 3, July–Sept. 2006, pp. 94–96, c3.
12. J. Jovanović and D. Gašević, "XML/XSLT-Based Knowledge Sharing," *Expert Systems with Applications*, vol. 29, no. 3, 2005, pp. 535–553.
13. Z. Obrenovic, D. Starcevic, and B. Selic, "A Model-Driven Approach to Content Repurposing," *IEEE Multimedia*, vol. 11, no. 1, 2004, pp. 62–71.
14. A. Eliëns, *DLP: A Language for Distributed Logic Programming: Design, Semantics, and Implementation*, John Wiley & Sons, 1992.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.

About the Authors

Željko Obrenović is a senior researcher at the Centre for Mathematics and Computer Sciences (CWI) in Amsterdam. His research interests include human-computer interaction, software engineering, the Semantic Web, and service-oriented architectures. He received his PhD in computer science from the University of Belgrade. Contact him at Kruislaan 413, 1090 SJ Amsterdam, The Netherlands; zeljko.obrenovic@cwi.nl.

Dragan Gašević is an assistant professor in the School of Computing and Information Systems at Athabasca University. His research interests include the Semantic Web, model-driven software engineering, knowledge management, service-oriented architectures, and learning technologies. He received his PhD in computer science from the University of Belgrade. He's a member of the IEEE Computer Society and ACM. Contact him at SCIS, Athabasca Univ., 1 University Dr., Athabasca, AB T9S 3A3, Canada; dgasevic@acm.org.

ADVERTISER INDEX SEPTEMBER/OCTOBER 2007

Advertiser	Page Number
Classified Advertising	25
East Carolina University	14
EclipseWorld 2007	Cover 3
ESRI	20
ICSQ 2007	1
Seapine Software, Inc.	Cover 4
Software Test & Performance Conference 2007	4
Starwest 2007	Cover 2

Advertising Personnel

Marian Anderson
Advertising Coordinator
Phone: +1 714 821 8380
Fax: +1 714 821 4010
manderson@computer.org

Sandy Brown
IEEE Computer Society,
Business Development Mgr
Phone: +1 714 821 8380
Fax: +1 714 821 4010
sb.ieeemedia@ieee.org

Advertising Sales Representatives

Mid Atlantic (product/recruitment)
Dawn Becker
Phone: +1 732 772 0160
Fax: +1 732 772 0164
Email: db.ieeemedia@ieee.org

New England (product)
Jody Estabrook
Phone: +1 978 244 0192
Fax: +1 978 244 0103
Email: je.ieeemedia@ieee.org

New England (recruitment)
John Restchack
Phone: +1 212 419 7578
Fax: +1 212 419 7589
Email: j.restchack@ieee.org

Connecticut (product)
Stan Greenfield
Phone: +1 203 938 2418
Fax: +1 203 938 3211
Email: greenco@optonline.net

Southwest (product)
Steve Loerch
Phone: +1 847 498 4520
Fax: +1 847 498 5911
Email: steve@didierandbroderick.com

Northwest (product)
Peter D. Scott
Phone: +1 415 421-7950
Fax: +1 415 398-4156
Email: peterd@pscottassoc.com

Southern CA (product)
Marshall Rubin
Phone: +1 818 888 2407
Fax: +1 818 888 4907
Email: mr.ieeemedia@ieee.org

Northwest/Southern CA (recruitment)
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Midwest (product)
Dave Jones
Phone: +1 708 442 5633
Fax: +1 708 442 7620
Email: dj.ieeemedia@ieee.org
Will Hamilton
Phone: +1 269 381 2156
Fax: +1 269 381 2556
Email: wh.ieeemedia@ieee.org
Joe DiNardo
Phone: +1 440 248 2456
Fax: +1 440 248 2594
Email: jd.ieeemedia@ieee.org

Southeast (recruitment)
Thomas M. Flynn
Phone: +1 770 645 2944
Fax: +1 770 993 4423
Email: flynntom@mindspring.com

Midwest/Southwest (recruitment)
Darcy Giovingo
Phone: +1 847 498-4520
Fax: +1 847 498-5911
Email: dg.ieeemedia@ieee.org

Southeast (product)
Bill Holland
Phone: +1 770 435 6549
Fax: +1 770 435 0243
Email: hollandwfh@yahoo.com

Japan (recruitment)
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Europe (product)
Hilary Turnbull
Phone: +44 1875 825700
Fax: +44 1875 825701
Email: impress@impressmedia.com