ITEA
INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

sirena

INNOVATION REPORT

The explosive growth of service-oriented architecture adoption

Although service-oriented architecture (SOA) is normally implemented for a business-processing environment, the ITEA project SIRENA applied it successfully in an environment with embedded devices. This document was produced as a contribution to the ITEA programme report 2005 (chapter 'Developments in the programme environment').

An explosive growth of SOA adoption

As some analysts predicted, e.g.:

"If you don't yet know about SOA and how it will change your enterprise's IT architecture, you are placing yourself at a competitive disadvantage. By 2008, SOA will be a prevailing software engineering practice, ending the 40-year domination of monolithic software architecture (0.7 probability)."
(Yefim Natis, Gartner, 2003)

2005 saw a sharp rise in the adoption of the service-oriented architecture paradigm:

"We can probably characterize 2005 as the year SOA went from being some foggy notion inside a crystal ball to being a design pattern that gained wide acceptance."
(Dana Gardner, Interarbor Solutions, 2005)

"Everybody I'm now talking to is working on SOA. People have at least an inkling of what it means and they're starting to plan it."
(Anne Thomas Manes, Burton Group, 2005)

"If you're not doing SOA, you're in serious danger. Every sizable software vendor has stated its future roadmap is going to be SOA related. If you don't adopt SOA, you could be cutting yourself off and not be able to upgrade your current applications."
(Ron Schmelzer, ZapThink, 2005)

"SOA is heading toward broad implementation in only a matter of a few years in the United States, regardless of organization size or vertical industry. 2006 will be the year of initial SOA project completion on a broad basis: not on a hit-or-miss trend, but through a rising tide of broad and deep adoption of SOA across the market."
(Tom Dwyer, Yankee Group, 2005)

Indeed, the SOA concept has gained significant hold in just a few years and will undoubtedly have a strong impact in many branches of technology. This tendency will certainly continue in 2006, especially in conjunction with the planned release of Microsoft's major new Windows platform, code-named Longhorn, of which the client platform – named Windows Vista – was first released in 2005. The Windows Communications Framework – formerly code-named Indigo and one of the Longhorn/Vista pillars – will be a powerful enabler for SOA implementation. All other major software editors – IBM, BEA, Sun, HP, Oracle, SAP, etc. – are moving in a similar direction; SAP deserves credit for having undertaken to overhaul its monolithic enterprise resource planning (ERP) platform, evolving it into the modular, service-based NetWeaver technology as part of its Enterprise Services Architecture.

INNOVATION REPORT

Service-oriented architecture in a nutshell

There are many definitions of the SOA concept [1]. For the purpose of this report, the following – definitely incomplete – definition may suffice:

A service-oriented architecture is a set of architectural tenets for building autonomous yet interoperable systems.

This definition includes two key words: *autonomous* and *interoperable*. The principal characteristics setting apart *autonomous* systems are that:

- They are created independently of each other;
- They operate independently of their environment; and
- They provide self-contained functionality, i.e. this functionality would be useful even if it was not associated with any higher-level systems.

Interoperability is favoured by clearly separating the interface that a service exposes to its environment from the implementation of that service.

Autonomy and interoperability are contradictory properties. One of the challenges of SOA is, therefore, to reconcile these opposing principles. The service-centric architectural principles for addressing this challenge can be summarised as follows:

- The design of a service interface follows an outside-in approach: rather than deriving from the details of its implementation, the service interface is defined by focusing on how the service may fit in a larger business-process context. Indeed, SOA is business-process centric rather than technology centric – a service usually represents a business task. Correspondingly, a service interface is mostly coarse-grained and stateless, and is based on the exchange of messages and documents. This aspect differentiates service-oriented from object-oriented design; the latter usually involves a conversation-style interaction pattern addressing individual object attributes.

- In an SOA, communicating entities are loosely coupled, by virtue of the fact that a service's functionality is exposed at its boundary, in terms of service 'contracts' (interfaces) together with schemas describing the messages and documents exchanged in a standardised, platform-neutral format. Thus, the implementation of the service may be modified without the service's users being affected. In contrast, the tightly coupled, context-related and stateful interactions found in distributed-object architectures create artificial dependencies between the communicating entities, resulting in rigid and fragile systems.

- In an SOA, communications are based on asynchronous message exchanges: when an action is invoked on a service, the result – if any – is returned to the invoking application entity without the latter suspending its operations. This is a departure from the synchronous remote procedure call paradigm that has proven to be much less scalable, especially in the presence of complex business processes where many operations with variable response times run concurrently.

[1] E.g., 'A defining moment for SOA', published by SearchWebServices.com in October 2004: http://searchwebservices.techtarget.com/originalContent/0,289142,sid26_gci1017004,00.html, complemented by: http://searchwebservices.techtarget.com/original/Content/0,289142,sid26_gci1044083,00.html in January 2005.

- In addition to its contract, a service can have requirements, properties or capabilities – expressed through 'policies' – that can be negotiated at runtime. This is particularly useful for specifying characteristics such as quality-of-service (QoS) level, security support and performance requirements. The capability to negotiate their use at configuration time helps preserve the service's generic property and re-usability.

The above architectural tenets are inter-related: richer content enables looser coupling, which in turn enables asynchronous communication.

Although SOA can be implemented in various ways, there is no doubt that Web Services technology will be the preferred implementation vehicle, in line with the prediction by Gartner in April 2003:

"SOA does not require Web Services but Web Services are based on accepted standards and will drive SOA to the mainstream. Through 2008, SOA and Web Services will be implemented together in more than 75% of new SOA or Web Services projects (0.7 probability)."

This almost universal adoption of Web Services technology is largely due to its total neutrality with regard to implementation platforms.

SOA in the device space

While the quotations above on the adoption of SOA relate primarily to the business-processing environment, this tendency can be expected to also produce a strong impact in the device space.

Today's Internet is connecting millions of people but, in the near future, Internet technology will network-connect billions of devices. As on-going miniaturisation allows embedded devices to become full-blown data-processing engines, intelligent networked devices of all kinds are emerging, even at the lowest level of the device hierarchy, i.e. sensors and actuators.

The ever-increasing power put into embedded devices can be harnessed to service-enable them, using the same SOA paradigm and associated Web Services technology as for business-process environments.

This, in turn, opens unprecedented perspectives of enabling seamless integration of device-level functionality into higher-level business processes. Ultimately, by analogy with existing examples such as human society and biological organisms, it should be possible to create a society of Web Services where components of all kinds, including devices, collaborate with each other to achieve their own individual goals.

The ITEA SIRENA project [2], led by Schneider Electric, anticipated this evolution three years ago and has demonstrated the feasibility of implementing SOA using Web Services in low-level devices with current technology. Indeed, it realised a proof-of-concept implementation of the *Devices Profile for Web Services* (DPWS) [3], destined to be integrated into embedded devices for a variety of applications in the industrial automation, home, automotive and telecommunications sectors. It implements all the DPWS protocols except WS-Security.

[2] See: http://www.sirena-itea.org.

[3] See: http://specs.xmlsoap.org/ws/2005/05/devprof/devicesprofile.pdf.

INNOVATION REPORT

This *DPWS Toolkit* – written in C – was derived from the gSOAP open-source software package, which has been modified to accommodate asynchronous control flows based on the Simple Object Access Protocol (SOAP) and WS-Addressing. It is fed by a Web Services Description Language (WSDL) file, from which it generates C structures representing the message contents, together with marshalling/demarshalling code for transforming between C structures and SOAP messages, as well as proxy client-side and skeleton server-side code. It has been ported to several target software platforms: Linux, Microsoft's Windows and Windows CE, Sun's Solaris, ExpressLogic's ThreadX and Quadros Systems' Quadros. At present, it runs on various hardware platforms, including ARM7- and Intel XScale-based processor boards, as well as a single-chip ARM7-based component.

Based on these hardware and software components, several experimental devices have been built, connected through 100-Mbps Ethernet networks. This network transfer rate is so high that transmission delays are almost negligible compared with CPU processing times. Early measurements on an ARM7-based platform running the ThreadX operating system (OS) show the following results:

- The static memory footprint of the device software including the OS, the TCP/IP protocol stack and the DPWS software is less than 500 Kbyte, while the dynamic memory requirements are below 100 Kbyte.
- The total time required for preparing and sending a request message to a device and for receiving and handling its response message is about 39 ms.

On the XScale-based platform, the measured aggregate request and response handling time is about 10 ms.

Since this DPWS implementation purported to demonstrate the feasibility of incorporating Web Services in embedded devices, it is not yet an industrial-strength product and there is ample room for improving its performance characteristics. Directions being explored include:

- Use of faster processors;
- Software optimisations – better memory management, faster string manipulation, etc.;
- Using SOAP directly on top of TCP/IP; and
- Using binary instead of textual encoding of SOAP messages.

Overall, it is estimated that an improvement of between one and two orders of magnitude over the currently observed processing speed is achievable. Thus, while sub-millisecond message processing times can easily be obtained on personal computer (PC) class devices today, this remains a challenge on embedded devices. This goal seems to be within reach, however, and, when attained, will enable use of DPWS in a wide range of devices for many types of applications.

The DPWS Toolkit is the property of Schneider Electric, but can be made available to third parties for experimental use. It participated successfully in the first DPWS Interoperability Workshop that took place in October 2005.

In the near future, Schneider Electric will industrialise the toolkit and integrate it into several of its industrial automation devices. Planned extensions – in addition to those mentioned in relation to performance improvements – include:

- Support of IPv6;
- Support of WS-Security;
- Support of WS-ReliableMessaging;
- Implementation of the discovery proxy functionality allowing scaling out the device discovery process; and
- Code generators for Java and C#.

INNOVATION REPORT

The next major move will be to implement a comprehensive, scalable, easy-to-deploy SOA ecosystem for devices, i.e. a complete tool suite for design, development, deployment and runtime support. This includes aspects such as service aggregation, composition and orchestration, and is fully in line with analyst forecasts such as:

“2006 will be the year of SOA automation methods, tools, and infrastructure.”
(Dana Gardner, Interlabor Solutions, December 2005)