Web Services on Embedded Devices

Daniel Schall
d.schall@infosys.tuwien.ac.at
Distributed Systems Group, TUWien
Argentinierstrasse 8, 1040 Wien, Austria
Marco Aiello
aiellom@ieee.org
Department of Information and Telecommunication Technologies
University of Trento, Via Sommarive 14, 38050 Trento, Italy
Schahram Dustdar
dustdar@infosys.tuwien.ac.at
Distributed Systems Group, TUWien
Argentinierstrasse 8, 1040 Wien, Austria

Received: January XX 2005; revised: November XX 2005

***Abstract*— The capabilities of embedded devices such as smartphones are steadily increasing and provide the great flexibility of data access and collaboration while being mobile. From the distributed computing point of view, fundamental issues in mobile computing include heterogeneity in terms of varying device capabilities (i.e., operating systems and various hardware platforms), performance characteristics and real-time behavior, and the ability to discover and interact with peers seamlessly. Web services are a family of XML based protocols to achieve interoperability among loosely coupled networked applications. We propose the use of Web services on embedded devices in order to solve interoperability issues in distributed mobile systems. We discuss various toolkits available for embedded devices and investigate performance characteristics of embedded Web services on smartphones. Our goal is to guide the design of Web services based applications on mobile devices, and provide estimates of performance that can be expected.**

***Index Terms*— pervasive computing, mobile devices, web services, performance study**

I. INTRODUCTION

Web services are a family of XML-based standards designed for the communication of loosely coupled, dynamically bound applications. Web services can be seen as the evolution of the Web where not only humans interact with applications via HTML forms, but also applications interact directly with each another. Alternatively, Web services can be seen as a generalization of distributed object middleware, such as CORBA [24]. In whichever way one considers Web services, there is a strong trend and wide adoption of the technology as means for addressing interoperability issues.

Web services are not only a recent technological hype, but are also defining a new way for designing applications and information systems. They are the most known incarnation of the programming paradigm known as Service-Oriented Computing [18], which has the goal of enabling Service-Oriented Architectures (SOA, e.g., see [4]). Of course, interoperability comes at a price. Most notably, the use of XML for the communication is computationally expensive and bandwidth consuming. Furthermore, using XML does not solve per se the ontological problem of having independent applications interoperate. Nevertheless, Web services are ready to penetrate embedded systems and mobile devices, and solve the interoperability problem [2].

In a complex setting of devices and embedded systems a number of issues need to be considered: on the one hand, heterogeneity and interoperability, determined by the device's functional properties (e.g., static hardware capabilities made available through a descriptive interface), and on the other hand, continuously changing capabilities (e.g., due to various operational conditions such as CPU utilization and memory usage), which limit the ability to contribute to a task at hand [12]. A task may require composition of a set of devices and aggregated functionality in order to meet task specific requirements. One important aspect is to find a feasible device configuration (e.g., interoperability among heterogeneous devices) in order to satisfy task specific needs, the other challenge is to execute a task in a satisfactory fashion (e.g. according to task dead-lines or QoS agreements [16]).

Web services provide an ideal framework to address issues such as heterogeneity and interoperability. However, it is of major importance to understand the performance limits and constraints, in terms of resource requirements, imposed by various Web services toolkits, in order to estimate expected performance at run-time (i.e., executing a task at hand).

We propose to use Web services as enabling infrastructure for the interoperation of heterogeneous mobile devices. In particular, we study the performances of various solutions for realizing Web services on mobile devices.

The remainder of this paper is organized as follows. In Section II, we describe potential application scenarios. Section III is a review of a number of Web services toolkits available for mobile devices and outlines design and performance issues. A performance study of embedded Web services is given in Section IV, including a performance evaluation. Related work and projects are discussed in Section V. Finally, Section VI draws conclusions on our performance study and Web services on mobile devices.

II. APPLICATION SCENARIOS

There are many scenarios in which heterogenous mobile devices need to interoperate. For instance, application scenarios in the mobility domain and in the home domain, where we find an increasing number of embedded devices with network capabilities.

A. MOBILE APPLICATIONS AND SERVICES

Wireless networks with varying coverage (e.g., 3G, WiFi, etc.) provide data access while on the move. In essence, the type of mobile application can be categorized by the data traffic. According to UMTS QoS traffic classes, we distinguish between 1) conversational, 2) streaming, 3) interactive, and 4) background classes [5].

The following sections introduce typical applications and services that can be found in aforementioned traffic classes. The distinct characteristic is real-time versus non real-time data traffic.

1) Real-Time Multimedia: Multimedia content such as real-time audio and video fall into the conversational or streaming class. The Session Initiation Protocol (SIP) [13] and the Real-Time Transmission Protocol (RTP) [14] provide the negotiation/transport framework to deliver delay sensitive A/V multimedia content. Various architectures, for instance situated devices and resources [20], may be used to overcome limits of mobile devices (e.g., display size, computational power, etc.) and deliver media to mobile users in an optimized fashion.

2) Interactive Mobile Multimedia Applications: In contrast to real-time multimedia applications, Web services based applications follow the request/response pattern and thus are set up in the interactive class i.e., best effort.

In general, we categorize mobile applications, based in the interactive traffic class, in information *consumer* and *provider*. The former includes information access through search queries, document download, etc. performed by mobile users; whereas the latter refers to contents or data that is provided or shared on mobile devices, for instance captured multimedia content such as still images, audio files, etc.

Location based services are among the first mobile applications that have been developed by the ubiquitous computing community [11]. Application scenarios include localized queries (e.g., finding restaurants in proximity) or providing information to the mobile user through location based push [9] (e.g., information that is customized based on user's position determined by GPS coordinates). For example, the Google Web services API [10] provides access to the Google search engine from mobile devices and thus can be used to obtain localized information, if combined with a form of localized search.

Location information is only a subset of the user's context. Various architectures and systems have been proposed to collect and aggregate sensory data and infer high level contextual information (a survey on context aware systems is given in [3]).

B. NETWORKED DEVICES AND APPLIANCES AT HOME

Domotics is the field where housing meets technology in its various forms (informatics, but also robotics, mechanics, ergonomics, and communication) to provide better homes from the safety and comfort point of view. Traditionally, domotic solutions were provided by a single vendor, using proprietary solutions for communication, mostly closed and expensive. This is no longer true. Domotic elements are heterogeneous in all aspects. Devices come from various vendors, have different hardware capabilities, network interfaces, operating systems, and yet need to have the ability to interoperate. Users need to have a unique view on all hardware elements and devices located at their homes. In addition, various tasks such as self-configuration should be done automatically, through internal communication and coordination, without requiring human intervention or manual configuration.

In order to solve the complexity of the home distributed system and to allow cooperation among the set of independent devices, it is necessary to have standard protocols and ways of communicating among them. Web services are an enabling technology for this task, as proposed in [2]. The solution to the interoperability comes at the cost of computational resources for handling XML messages, though, the growth in computational power, communication abilities, battery life of the devices, together with the lowering of the prices makes the approach evermore feasible.

III. WEB SERVICES ON EMBEDDED DEVICES

Implementing the Web service stack on mobile devices is today reality. In particular, there are a number of Web services toolkits available for mobile devices, in C++ as well as Java/J2ME (e.g., Symbian based devices), and furthermore .NET implementations on Microsoft platforms. Since those devices have constraint resources, i.e., CPU, memory and battery, the choice of the toolkit is crucial for the application's performance.

A. WEB SERVICES TOOLKITS

1) C++ TOOLKITS: gSOAP is a platform independent toolkit for Web services [8], which includes a WSDL parser `wsdl2h` (creates header files) and a stub/skeleton compiler `soapcpp2`. Depending on the SOAP client/server requirements, *.c* or *.cpp* files can be generated. The run-time library, `stdsoap2`, serializes and de-serializes calls and is the only dependency needed on the target platform. Figure 1 illustrates the gSOAP (client) run-time and shows the development and deployment cycle. The development process starts with C/C++ header file creation based on the

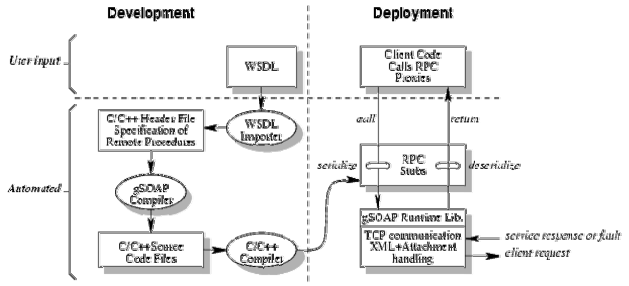

Fig. 1. gSOAP WS run-time

service's WSDL file. Next, the gSOAP compiler is used to create the code files. At run-time (i.e., the deployment) RPC calls are made on client side proxies [8].

2) TOOLKITS FOR J2ME: The J2ME platform (Java 2 Platform Micro Edition) is a set of standard Java APIs defined through the Java Community Process (JCP). The J2ME specifications define the Connected Device Configuration (CDC) (i.e., a subset of J2SE) and the Connected Limited Device Configuration (CLDC). In contrast to CDC, CLDC provides libraries such as the *Connection Framework* which are suitable for devices with a small memory footprint (not part of J2SE). CLDC targets hardware platforms with 128 KB to 512 KB memory and 16-bit or 32-bit CPUs. The Mobile Information Device Profile (MIDP) is specifically designed for cell phones and provides the user interface, network connectivity, local data storage, and application management needed by these devices.

Following SOAP APIs and Web services toolkits are suitable for J2ME/MIDP based devices.

1) kSOAP is an open source SOAP API for J2ME devices [1]. It provides a lightweight way to access SOAP based Web services. However, kSOAP cannot generate client side stubs from the web service's WSDL file.
2) JSR-172 is a set of Web services APIs (WSA) for J2ME [22] available in Sun's wireless tool kit (WTK) 2.2. In contrast to kSOAP, client side stubs can be generated using WSDL files, which accelerates the development process. WSA for J2ME has thus similar capabilities (e.g., stub generator) compared to gSOAP's client runtime. However, it is important to note that WSA is only suitable for consumption of Web services on mobile devices (i.e., it cannot provide a service on a mobile host).

3) WEB SERVICES FOR .NET COMPACT FRAMEWORK: The .NET Compact Framework (CF) is a subset of Microsoft's .NET framework. The .NET CF is supported on various devices/platforms that are based on PocketPC and Smartphone architectures. Web services on .NET CF support the use of synchronous or asynchronous invocation [17]. Development of embedded Web services is analog to implementing Web services clients in .NET. A *Web Reference* (i.e., a reference to the actual service) has to be added to a project and code is automatically generated.

B. PERFORMANCE METRICS

In order to choose a tool for implementing the Web service stack on a mobile devices, we need to consider a number of parameters on which to establish such a choice. Following, we identify a number performance metrics and design considerations:

1) Average time needed to execute a given request.
2) The latency given a number of requests to be executed.
3) The maximum number of concurrent requests that can be executed.
4) Overhead of using Java (e.g., multi-threaded Java application) in terms of startup overhead, CPU usage and memory consumption (i.e., allocation given a number of requests to be executed).

In our following performance study, we focus on a client or Web services consumer scenario, however, discussions have analog significance for service providers implemented on mobile devices.

IV. A COMPARATIVE PERFORMANCE TEST

The aim is to compare the performance of C++ Web services with Java implementations using the Symbian OS. We evaluate performance, in terms of latency obtained through roundtrip delay measurements of SOAP/XML messages per second.

A. METHODOLOGY

Performance estimates can be obtained through analytical modeling and simulation or through an empirical study such as observed measurements (e.g., measuring roundtrip delay using time stamped messages). In addition, various Java profilers are available to analyze the performance and bottlenecks of Java applications at run time (e.g., memory overhead, function calls, etc.).

We use a black box approach and measure roundtrip delay obtained through time stamped messages as we have no knowledge of the server side configuration and load. In addition, we use the Java profiling tools available in Sun's WTK 2.2 to obtain "offline" information about the Java SOAP toolkit (i.e., Memory Monitor and Method Invocation Graph) [1].

A straight forward way to obtain packet statistics is to use the `ping` utility (using the ICMP protocol). For a given packet size, we measure the roundtrip delay (roundtrip time RTT) for each packet and some statistics such minimum, maximum and average RTT time in millisecond. For our experiments we take a similar approach and add time stamps to each SOAP request/response invocation. In order to trace a request, we add a time stamp t_1 at method invocation on the SOAP client, t_2 upon sending the SOAP message through the socket interface, t_3 upon receiving a response on the socket interface and finally t_4 when getting the result of the SOAP call. We can calculate following timer intervals: $T_{wstack}(t_2 - t_1)$, gives us the time needed to create a SOAP message (time spent in the web

[1]WTK profiling tools are available for the wireless terminal emulator. It is important to note that a Java profiler adds significant CPU load and memory overhead to observed system, and thus should not be used for a comparative online performance study.

services stack), $T_{network}(t_3 - (t_2 + \tau))$ time to receive a response on the socket interface and $T_{wstack}(t_4 - t_3)$, which is the processing time of the response message in the stack.

$T_{network}$ is governed by various random variables (e.g., actual network load). By using Java, we add a time-offset τ (Java socket to native socket interface) which will be neglected in our measurements.

B. EXPERIMENTAL SETUP AND IMPLEMENTATION

We use a Symbian OS v8.0a based device (S60 2nd Edition FP 2 developer platform) and invoke Web services requests using the Google Web services API [10]. The choice of the toolkit is thus limited to Java and C++. In particular, a problem arises when trying to consume the Google API with WSA for J2ME. The WSA stub generator tool, available in WTK 2.2., fails due to unsupported SOAP binding style (i.e., only *document* style is supported in WSA, however, the Google API uses *rpc* style).

IP connectivity can be provided through 2.5G/3G (GPRS or WCDMA) wireless networks or Bluetooth. In our tests, a Bluetooth connection between the mobile device and a Windows XP based PC (acting as a remote access server) was configured.

The goal of our architecture is to execute multiple SOAP requests concurrently. Therefore, we use multiple threads to achieve non-blocking operation. The native C++ Symbian API supports multi-threading through the `RThread` class. On J2ME, standard Java threads can be used to accomplish concurrent execution of requests.

In Section III-B we enumerated various metrics that should be considered at design time. The very basic task is to execute a single request, thus spawning a thread $\mathbf{t}$ for request r_i to be executed, a task tuple $(\mathbf{t}, r_i)$. However, the maximum number of requests that can be executed simultaneously (i.e. the number n of requests in execution $r_{exec}^{(t)}$, at particular time instance t) needs to be taken into account. In Java, we implemented a standard thread pool and limit the thread pool size by K (e.g., thread pool size $K = 4$)[2]. Next, we execute N requests in a batch manner. The time needed to execute N requests is determined by the life-cycle $T_k^{(n)}$ of a thread $\mathbf{t}_k$ (a life-cycle is calculated by $T_{cycle}(t_{end}^{(n)} - t_{start}^{(n)})$), where $k = 1...K$ is the thread index executing request $n = 1...N$.

C. RESULTS AND DISCUSSION

The results of the experimentation with the Google APIs are summarized in Figure 2. In the figure, a direct comparison of C++ and Java for a set of request rates is presented. The diagram shows average time values for T_{send}, time interval to create a request, $T_{receive}$, time interval to receive the corresponding response, $T_{network}$, the network latency, T_{req}, total time required to execute a given request and receive response, and finally T_{total} which is the total time needed to process all requests and responses.

[2]If we set $K > 8$, Java throws a Symbian exception, error `KErrServerBusy`. This error is caused by `RSessionBase`, as a result of the server being busy handling another request.

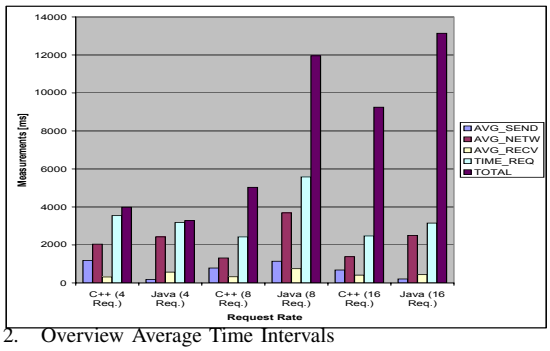

Fig. 2. Overview Average Time Intervals

As illustrated in Figure 2, Java performs better in creating a request T_{send}, but poorly in receiving a response $T_{receive}$. Slow SOAP response processing becomes more apparent when executing the Java application on the WTK emulator [3]. In average, the C++ toolkit is faster in executing a number of requests T_{total} and faster in executing a given request T_{req}.

Figure 3 illustrates best, worst and average processing time, given a number of requests to be executed. In this diagram net processing time in the Web services stack is shown (i.e., we neglect $T_{network}$).

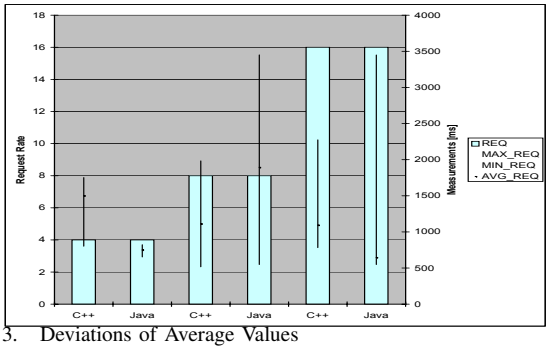

Fig. 3. Deviations of Average Values

Java has in general larger deviations in `min`/`max` values. Reasons are non-deterministic garbage collection on the Java Kilobyte Virtual Machine (KVM) and code optimization at run-time (i.e., optimization of repeatedly executed parts of the code, called hotspots). This gives up to 50% faster execution of subsequent requests, compared to the first ones (e.g., the first K executions).

Table I shows obtained performance metrics (values are rounded).

TABLE I
PERFORMANCE METRICS RESULTS

Toolkit	Max Req.	Avg Req.[ms]	8 Req.[ms]	16 Req.[ms]
gSOAP	> 16	2800	5000	9000
kSOAP	≤ 8	3800	12000	13000

The maximum number of concurrent requests is significantly higher using gSOAP/C++. However, in order to obtain

[3]The profiler's method invocation graph shows a main bottleneck in processing `SoapSerializeEnvelop.readUnknow` in the kSOAP stack.

comparable measurements (e.g., time needed to execute multiple requests), we limited the maximum number of requests in execution in C++ as well (i.e., K set to 4).

V. RELATED WORK

Application areas of collaborative embedded devices span diverse research fields. Our research is related to embedded Web services and performance characterization. Understanding limits and characteristics are significant in various application domains where time critical information and real-time monitoring are important.

Attention is needed in the area of aging in place as the number of the elderly population increases [21]. Physical inabilities due to aging and cognitive decline yield the need to monitor health status, and require infrastructures for home health care. Domotic technologies provide appropriate solutions for health care at home. The goal is to provide a level of comfort and safety, and let the elderly live in his/her own home without invading everyday activities or the environment. Thus, transparent and intelligent systems are needed to aid the user in his/her needs.

Work in the area of interoperability problems and SOA, with special focus on legacy issues, can be found in Jammes et al. [15]. The work involves the implementation of a Web services stack on devices following a form of device profiling. The stack proposed is based on WS-Eventing and implemented on Linux, Windows, Windows CE, ThreadX, and QuadrOS. QoS levels of the stack, such as WS-Security, are not implemented.

From the technological point of view, a performance study of SOAP based servers on mobile devices is conducted by Pham and Gehlen [19]. A mobile server using the kSOAP API for J2ME is implemented, and a study and analyzes of J2ME multi-threading behavior is presented.

Engelen introduces the gSOAP toolkit in [7] and studies the performance of gSOAP (results available at [6]). The gSOAP toolkit is compared to .NET, Apache Axis Java and C++, and xSOAP. The goal of this performance study was to compare performance on standard PCs.

Tierno and Campo [23] provide an analyzes of the limits of Java applications on J2ME. The ability to process multimedia data, i.e. applications such as image processing, movement detection, and pattern recognition, on smart phones is discussed. In addition, various code optimization techniques, employed by KVMs, are highlighted.

VI. CONCLUSION

As high speed 3G networks such as UMTS and metropolitan WiFi are increasingly deployed, ubiquitous data access is becoming a reality. In addition, a large number of embedded systems (e.g., sensors and actuators) at home are equipped with wireless interfaces and may organize themselves autonomously. Web services on embedded devices address heterogeneity and enable interoperability among disparate systems. In this paper, we discussed various application scenarios in both domains. Considering application performance is a key to successful deployment of such systems on a larger scale, in order to meet task specific requirements.

We provided an overview of different Web services toolkits available for the Symbian platform. Our empirical study compares the performance of Java/J2ME based Web services with toolkits available in C++.

We discussed a framework to obtain online performance measurements, unobtrusively, through time stamped messages, e.g., at various layers in the Web services stack. These measurements included creation time of SOAP messages, processing time of messages, etc. This method provides a way to trace messages in the system and corresponding time spent at various layers.

We believe that Web services on embedded devices provide great flexibility and interoperability. Web services play an important role in accessing or providing multimedia content on mobile devices. They aid the process of sharing multimedia files that may be captured in situ, and also help in locating content in distributed systems such as large scale video archives.

REFERENCES

[1] SOAP Implementation for J2ME. `http://kobjects.sourceforge.net/`.

[2] M. Aiello. The Role of Web Services at Home. In *IEEE Web Service-based Systems and Applications (WEBSA)*, 2006. To appear.

[3] M. Baldauf, S. Dustdar, and F. Rosenberg. A Survey on Context Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2006. To appear.

[4] D. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[5] S. Dixit and R. Prasad. *Wireless IP and Building the Mobile Internet*. Artech House Publisher, 2003.

[6] R. A. Engelen. *SOAP/XML Web Service Performance*. `http://www.cs.fsu.edu/~engelen/soapperformance.html`.

[7] R. A. Engelen. Pushing the SOAP Envelope with Web Services for Scientific Computing. *The International Conference on Web Services ICWS*, 2003.

[8] R. A. Engelen. gSOAP: C/C++ Web Services Toolkit, 2004. `http://gsoap2.sourceforge.net/`.

[9] I. Miladinovic G. Pospischil, J. Stadler. A Location-Based Push Architecture using SIP. In *4th International Symposium on Wireless Personal Multimedia Communications (WPMC 2001)*, September 2001.

[10] Google. Google API Specifications. `http://www.google.com/apis/`.

[11] J. Hightower and G. Borriello. A Survey and Taxonomy of Location Systems for Ubiquitous Computing, August 2001.

[12] B. Horan. The Use of Capability Descriptions in a Wireless Transducer Network. Technical Report TR-2005-131, Sun Microsystems, 2005.

[13] IETF. Session Initiation Protocol (SIP). `http://www.ietf.org/html.charters/sip-charter.html`.

[14] IETF. RTP: A Transport Protocol for Real-Time Applications, January 1996. `http://www.ietf.org/rfc/rfc1889.txt`.

[15] F. Jammes, A. Mensch, and H. Smit. Service-Oriented device communications using the devices profile for web services. In *MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–8. ACM Press, 2005.

[16] A. Lazovik, M. Aiello, and M. Papazoglou. Planning and Monitoring the Execution of Web Service Requests. In *Conf. on Service-Oriented Computing (ICSOC-03)*, Lecture Notes in Computer Sciences 2910, pages 335–350. Springer, 2003.

[17] Microsoft Developer Network (MSDN). Consuming Web Services with the Microsoft .NET Compact Framework, March 2003. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetcomp/html/netcfwebservices.asp`.

[18] M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Commun. ACM*, 46(10):24–28, 2003.

[19] L. Pham and G. Gehlen. Realization and Performance Analysis of a SOAP Server for Mobile Devices. volume 02, pages 791–797. VDE Verlag, Apr 2005.

[20] T. L. Pham, G. Schneider, S. Goose, and A. Pizano. Composite Device Computing Environment: A Framework for Augmenting the PDA Using Surrounding Resources. In *Workshop on Situated Interaction in Ubiquitous Computing at CHI 2000*, April 2000.
[21] V. Stanford. Using pervasive computing to deliver elder care. *IEEE Pervasive Computing*, 1(1):10–13, 2002.
[22] Sun Microsystems. J2ME Web Services Technical White Paper, July 2004. `http://java.sun.com/j2me/reference/whitepapers/Web_Svcs_wp072904.pdf`.
[23] J. Tierno and C. Campo. Smart Camera Phones: Limits and Applications. *IEEE Pervasive Computing*, 04(2):84–87, 2005.
[24] S. Vinoski. CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14(2), 1997.