

Software Design - Position Paper: SOA on Wireless Networks

Pat Freestone {pgf23@drexel.edu}
Steve Prazenica {stevepraz@gmail.com}
Kyle Usbeck {kfu22@drexel.edu}

November 13, 2007

1 Introduction

The Service Orientated Architecture (SOA) has become popular for implementing distributed computing applications. There are several benefits that SOA provides to these applications, such as interoperability, loose coupling, and componentization. These advantages are present due to the following characteristics of SOA applications [6]:

- Discoverability and Dynamic Binding
- Self-Contained and Modular
- Coarse-Grained Interfaces
- Self-Healing
- Location Transparency
- Composability

However, the benefits of SOA come at a cost. Although loose coupling and interoperability are often driving factors in software design, there are situations where these benefits are preempted by other concerns. For example, in hardware constrained and wireless network environments, a larger focus often needs to be placed on potential optimizations and performance.

This paper discusses the problems that arise from less-reliable network communications due to the disconnected nature of wireless networks in the context of SOA. Specifically, it argues that the high volume, large size, and format of transmissions typically associated with SOA make operating in wireless environments particularly problematic.

2 Discoverability and Dynamic Binding

One of the key characteristics of SOA is that services are discoverable and dynamically bound [8]. In this model a service consumer checks a third party registry at run-time for any service that it might need. The registry provides information in a standard format about any service that fulfills the consumer's need [6].

There are many benefits to architecting software in this manner, particularly in terms of loose coupling and interoperability. First, the dependency between the service consumer and the service provider is a runtime dependency rather than a compile-time dependency. Removing compile-time dependencies improves the maintainability of the application because a new interface binding is not required on the consumer side every time the provider's interface is changed [6]. Further, the third-party registry provides the only dependency between the provider and consumer, the contract [6].

However, the ease of maintenance and interoperability provided by this model does come at a cost, particularly in the area of performance. Developing applications using SOA requires the use of some standard protocols for service

discovery and description. The Simple Object Access Protocol (SOAP) is a common protocol used to facilitate this function, often in conjunction with the Web Services Description Language (WSDL) for web services [2].

SOAP attains a high level of interoperability by requiring that messages are sent using the Extensible Markup Language (XML) [2]. XML messages fit nicely into the SOA model because they provide broad platform support and interoperability options [7]. However, most XML-based devices place emphasis on the “plug-and-play flexibility” [4] of the communication without worrying about performance.

There are two main downsides to the use of XML in wireless network and hardware constrained application situations. These downsides are larger message size and slower encoding and decoding processes. Further, using such a generalized communication format ignores the optimizations available in homogeneous systems.

XML is designed to provide a structure around data in such a way that makes the data self-describing. This is done through the use of tags to describe the data within the message. Unfortunately, the start and end tags in XML can make up about half or more of the message’s physical size [2]. For applications that require high performance, binary messages are typically chosen due to a smaller size. The expansion of XML over binary for the same message is measured in various ranges, such as 4-10 times larger [2] and 6-8 times larger [4]. Binary encodings are particularly better when handling any form of opaque binary data as this often translates to a size increase of 33% or more for XML-based transmissions [7].

Another downside to the use of XML is the additional processing required for encoding and decoding the messages. The real cost of using XML in terms of encoding is the conversion to ASCII from binary and vice versa [4]. In particular, the most costly operations involved in this method are the conversion between ASCII and double [2]. Dynamic object generation is particularly taxing because it requires storing the schema definition as well as processing the document multiple times. Two common tools for parsing XML are the Document Object Model (DOM) and Simple API for XML (SAX). Both of these technologies require at least two passes through the XML message to construct the object model and interpret the underlying data [2].

Lastly, using such a generalized communication format ignores the optimizations possible on homogeneous systems. That is, if the services composing a system all use the same type of data. In such homogeneous systems, using binary communication protocols will add no cost above the network transport time [4], which is unavoidable in a distributed, networked application. Further, even in heterogeneous systems, studies have shown that the encoding and decoding times of XML are still much higher than binary protocols [4]. Because of the increased message size, higher encoding and decoding requirements, and the ignored optimizations of homogeneous systems, the benefits offered by discoverability and dynamic binding have drawbacks in wireless network and hardware constrained applications.

3 Self-Contained and Modular

One of the key characteristics of SOA is an architecture that is self contained and modular. This approach allows for code reuse and ease of maintainability by avoiding many vague dependencies in favor of a few well defined relationships. Code can be easier to maintain because the effects of changes are isolated to smaller areas. Also, it becomes possible to use modules in different configurations in order to achieve varied results without having to rewrite large amounts of functionality. These modules are separated in a way that aligns with the common business practices that the system must execute [6].

However, this modularity and self-contained nature does not come without a cost. The increased segmentation of the services can result in additional overhead. The coordination that must occur between modules because of the self-contained and modular nature of the services is not without an additional cost in resources [6]. There is an increased overhead as a result of having to coordinate all the self-contained, modules, much like the additional cost of coordinating the threads of a multi-threaded application.

Additionally, SOA modules are designed to interact over a network connection, even if they are running on the same physical machine [6]. There is a potential for optimizing the interaction that could be overlooked if all connections are network based. For instance, utilizing shared memory could replace operations such as communicating via network interfaces as well as object serialization and deserialization.

4 Coarse-Grained Interfaces

SOA is also geared towards coarse-grained rather than fine-grained interfaces. A fine-grained interface is typically used to perform a single, well-defined operation, which often leads to an increased number of network connections that must be established. On the other hand, coarse-grained interfaces typically complete multiple operations or return larger amounts of information. This results in fewer network connections but sometimes larger amounts data that must be transported. Coarse-grained interfaces are favored by SOA because of the problems and complexity that arise from a large group of fine-grained distributed objects [6]. With fine-grained interfaces, performance can suffer because of the large number of network operations and complex dependencies.

This is particularly true in wireless networks, where the nature of the medium and its potential unreliability do not lend themselves to constantly establishing large numbers of network connections. Also, the system becomes hard to maintain because of the numerous dependencies that exist. The SOA answer is to utilize coarse-grained interfaces to provide communication to the objects of a service. This also allows multiple coarse-grained interfaces to be used to organize services in different configurations, for different purposes [6].

However, there are varying degrees of granularity even among coarse-grained interfaces and services. This is an important decision that has to be made when considering the design of the interfaces and services. A service that is too coarse-grained will return more information than the user needs, potentially enough to be a security risk [6]. The larger data transfers are problematic in a wireless network application because malicious attackers have an easier time breaking certain types of security as the amount of packets transmitted increases. Coarse-grained interfaces could be returning more information than is used by the requestor, which is a waste of resources. On the other hand, a requestor may have to contact several fine-grained interfaces in order to get all the necessary information, which would require more network connections.

Thus, there are drawbacks with both fine-grained and coarse-grained interfaces in a wireless network application. Fine-grained interfaces require multiple network connections to accomplish one task while coarse-grained interfaces ignore the potential optimizations of custom-tailored interfaces. In relation to unreliable network connectivity, coarse-grained interfaces can be problematic because of the large transmission size. Specifically, the larger transmission will increase the probability that the entire transmission will be lost, thus requiring a complete retransmission.

5 Self-Healing

Another attribute of SOA is self-healing. Self-healing refers to “a systems ability to recover from errors without human intervention during execution” [6]. In particular, this characteristic encompasses how the system handles the presence or absence of other services as well as hardware failures [6]. The feature of self-healing is built into the service discovery protocols used by many SOA applications [3]. However, this robustness comes at a cost of increased overhead to the system as a whole.

There are several actions that an application must take to be self-healing. First, the application must implement failure detection techniques. These techniques can include monitoring periodic announcements and implementing bounded retries [3]. The application must also implement consistency-maintenance mechanisms to keep services aware of the availability of resources. Techniques to accomplish this task include notification and polling [3]. A third operation that a self-healing application must perform is recovery. This is often performed through application-level persistence or soft-state signaling, in which services must periodically update their status and if they do not, the service description is discarded until the service is again available [3]. Lastly, in an environment where certain services are mission critical, they must redundantly exist to provide failover availability for the application.

All of these features add overhead to the application, which can be very costly in a hardware-constrained wireless network application. First, techniques such as notification and polling require a larger number of network transmissions. Further, all of these features add an extra level of complexity for processing the metadata associated with each self-healing function. Lastly, in applications where redundant services are provided mechanisms must be used to choose between those resources.

6 Location Transparency

Location Transparency is an attractive feature of SOA because it decouples client requests from service providers. By utilizing directory services, UDDI, and dynamic binding discussed in Section 2, SOA developers can address components and services in the same way regardless of the services' locations [6].

The benefits of uniform addressing are encountered during development, runtime, and maintainance. Location Transparency allows system developers to concentrate on business domain problems rather than implementation details. Also, separation of services allows for easy service replacement whether it be a bug-fix or upgrade. Arguably the greatest benefit of Location Transparency on wireless networks is the ability to support service redundancy [5].

Unfortunately, the benefits associated with Location Transparency also tend to incur overhead costs, especially in wireless network environments. We've already discussed in Section 5 how choosing between redundant services can add overhead to a service-based system. More importantly, however, the reliability of a service is dependent on the availability of a network link [5]. In a wireless network, network links are typically more volatile than in wired networks, so it is dangerous to rely on any remote service. Also, Section 2 discusses the overhead associated with the service discovery traffic. Finally, the use of Location Transparency eliminates the possibility of sharing data between services in an optimized fashion (e.g. shared memory). This lack of optimization capability can severely impact application performance on resource constrained devices in mobile wireless networks.

7 Composability

Imagine a myriad of services that each represent business processes and can all discover each other dynamically. These services are themselves useful, however, if they cannot interact, their potential is extremely limited. When the output of one service is compatible with the input of another service, the two are said to be composable [1]. Thus, composability is the assurance that two services will work together.

While the benefits of composability are clear, the implications may not be as obvious. To ensure that services are "speaking the same language," the messages are sent in an ASCII formatted XML message [1]. The associated overhead costs of XML messaging are discussed in Section 2. As previously mentioned, XML messaging increases interoperability at the cost of larger network transmissions and increased processing.

8 Conclusion

Although SOA provides many benefits as a software architecture, namely loose coupling and interoperability, there are situations in which the overhead associated with these benefits can lead to problems. One situation where these problems occur is in a mobile wireless network environment. These problems include:

- Larger Transmission Size
- Increased Processing
- Ignored Optimizations in Homogenous Systems

This paper discusses these drawbacks in relationship to the core characteristics of SOA applications.

References

- [1] Jorge Cardoso and Amit Sheth. *Semantic Web Services and Web Process Composition: First International Workshop, SWSWPC 2004, San Diego, CA, USA, July 6, 2004, Revised Selected Papers (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
- [2] Kenneth Chiu, Madhusudhan Govindaraju, and Randall Bramley. Investigating the limits of soap performance for scientific computing. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 246, Washington, DC, USA, 2002. IEEE Computer Society.

- [3] C. Dabrowski and K. Mills. Understanding self-healing in service-discovery systems. In *WOSS '02: Proceedings of the first workshop on Self-healing systems*, pages 15–20, New York, NY, USA, 2002. ACM.
- [4] Greg Eisenhauer, Fabian E. Bustamante, and Karsten Schwan. Native data representation: An efficient wire format for high-performance distributed computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(12):1234–1246, 2002.
- [5] Jeff Hanson and Ryan Ireland. The benefits of location transparency in an soa. <http://articles.techrepublic.com.com/5100-22-1049544.html>.
- [6] James McGovern, Sameer Tyagi, Michael Stevens, and Sunil Mathew. *Java Web Services Architecture*. 2003.
- [7] MSDN. Large data and streaming. <http://msdn2.microsoft.com/en-us/library/ms733742.aspx>.
- [8] Dokovski N., I. Widya, and A. van Halteren. Paradigm: Service oriented computing. Freeband AWARENESS D2.7b.