Designing Survivable Services from Independent Components with Basic Functionality

Andreas Dittrich, Jon Kowal and Miroslaw Malek

Institut für Informatik
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin
{dittrich,malek}@informatik.hu-berlin.de
mail@jonkowal.de

Abstract: Service-oriented architectures focus mainly on the automatic configuration of the attributes that describe the different layers involved in service communication and treat service instances monolithically - they either exist in the network which means that they are fully usable or they do not. This approach does not work well in environments where services are insufficiently dependable and the types of services used are not well known or standardized. This paper proposes a model to compose complex services from independent components with basic functionality that are organized as minimal services in the same service-oriented architecture. The approach promises to better handle run-time diagnostics and on-the-fly (re-)composition of service functionality in networks with highly dynamic capabilities.

Keywords: Survivability, Adaptivity, Composition

1 Introduction

Information and communication technology has become ubiquitous in the last two decades. The cheaper production cost and increased networking capabilities allow mobile embedded devices to pervade areas of computing that used to have fixed environments and help to make them more flexible and dynamic.

Because of the plethora of new devices entering traditional computer networks and the differences in their capabilities and non-functional properties such as availability and performance, there is the need for standards to configure the communication layers that are outside of the semantics of the services provided by those devices. A service client or a server is only interested in the description of the actual service usage. For the sake of simplicity every other detail of network communication should be transparent. Quite a few frameworks have emerged that try to define the different layers involved in service communication and offer protocols to automatically configure the layers in decentralized, dynamic networks. What these frameworks describe is commonly known as a *service-oriented architecture*.

For the purpose of connecting cheap and resource restricted systems the technologies used in a service-oriented architecture need to scale. Only minimal overhead over the requirements for taking part in the network is acceptable. It has been shown in [DK08] that the requirements proposed by the *Zeroconf working group* in [Wil02] and their implementation described in [CS05] provide a good compromise between functionality and hardware requirements and can be implemented with very little overhead.

Zeroconf essentially takes care of IP interface configuration, the translation between host names and IP addresses and the discovery of services. It does that using the well standardised concepts of *AutoIP*, *multicast DNS* and *DNS-Based Service Discovery* and thus only depends on the existence of Multicast IP and User Datagram Protocol (UDP). Following the findings of [DK08] it can generally be said that if a device is able to be a part of an IP network it is also able to be a part of a zeroconf network. This makes zeroconf an ideal framework for realizing service-oriented architectures in which the participating nodes are equipped with limited memory and processing power.

2 Problem statement

Zeroconf focuses mainly on the automatic configuration of the attributes that describe the different layers involved in IP based service communication. Service instances are treated monolithically - they either exist in the network which means they are fully usable or they do not. Their functionality in all its possible complexity is seen as a whole. This works well in environments where services are sufficiently dependable and the types of services used are well known and standardized.

But by using this approach it is difficult to quantify non-functional attributes of services that are of key importance in modern dynamic service networks like dependability, fault proneness, fault tolerance and resilience - all of them having impact on the *survivability* of a certain service. Guaranteeing redundancy is expensive if the failure of one component of the service equals the failure of the service as a whole and the replacement of those complex service instances can prove to be non-trivial. It is also difficult to monitor the state of service instances: Zeroconf provides pretty good mechanisms to detect the availability of service instances, however if those mechanisms are only used on complex services seen as single units a lot of their potential remains unused.

We believe that if a complex service can be described by an optimized composition of components with minimized functionality, those smaller components are easier to monitor, maintain and replace in case of failure. If the functionality is basic enough they might even be replaced by components that are used by different complex services but belong to the same class of functionality. Given m complex services realized by n components with basic functionality, can one determine the maximum number k of components that may fail while all m services can still be provided? How many components may fail so that a certain service can still be provided?

Models are needed to extract the different functionalities of complex services and to quantify their impact on the service functionality as a whole. This might lead to more dependable services that are cheaper to maintain and - using the techniques of a service-oriented architecture - can sustain their functionality longer without manual interaction. The resulting service-oriented architecture can be called a *survivability-oriented architecture*. As in web service composition [MM04], the appropriate methods for component composition must be used.

3 Proposed solution

We claim that every complex service can be divided into disjunctive components with minimized functionality. Those components - being instances of simpler services - can be modeled as regular service instances in a service-oriented architecture, e.g. Zeroconf. They automatically configure their network parameters and propagate their availability to the other nodes and thus benefit from the same techniques as the complex service instance they provide. The model of such a complex service is as follows:

$$S = \{\{s_1, s_2, ..., s_n\}, C\} \quad (1)$$

S is defined as the functionality of the complex service we want to describe while s_i being the functionality of the simple component i. C contains the composition rules for all the s_i that make up the service. As long as the service instance S is able to discover all its s_i in the network it can compose them via the rules C and is usable to service clients.

The simple components have a small set of functionalities and vary in their dependence on the context they operate in. The context sensitivity is crucial to determine the interchangeability of a certain component. If the functionality of a component provides is basic enough and if it is independent of the context it operates in, a component can be used to build many different complex services. We call this characteristic *service-agnostic* because the component itself operates the same way, independent of the complex service that leverages its functionality, it does not even need to "know" the service that uses it. In contrast, it can be said that if a component with basic functionality is in any degree dependent on the context it operates in - e.g. location, time, temperature, type of complex service that uses it, availability of special hardware - it cannot be trivially replaced by components of the same functionality. So we extend the description of complex service functionality in (1) as follows:

$$S_j = \{\{s_{1j}, s_{2j}, ..., s_{nj}\}, \{t_1, t_2, ..., t_n\}, C\} \quad (2)$$

The s_{ij} now describe the simple components that are context-sensitive, t_i are the service-agnostic components. Both of them compose the complex service functionality S by the rules defined in C. Service-agnostic components provide several key-advantages:

1. They can easily be replaced in case of failure as their functionality is only bound to their existence in the network. This makes redundancy trivial.

2. By their very nature of providing independent, basic abstract functionality they can be used by many services at the same time.

Thus, as long as there is at least one service-agnostic instance of a special type of service left, all the complex services relying on the functionality provided by that instance will remain functional. On the other hand, it is also possible to introduce new services to an already existing service network without deploying additional resources: New services could be composed only from service-agnostic components. This is illustrated in the following example:

$$\begin{aligned} S_1 &= \{\{s_{11}, s_{21}\}, \{t_1, t_2\}, C_1\} \\ S_2 &= \{\{s_{22}\}, \{t_1, t_3, t_4\}, C_2\} \\ S_3 &= \{\{\varnothing\}, \{t_1, t_2, t_4\}, C_3\} \end{aligned} \tag{3}$$

As long as we have at least one instance for the service-agnostic functionality t_1 all three services can remain functional. If s_{22} fails however, S_2 cannot maintain its functionality. Service S_3 on the other hand can be seen as a new service. In an already working service network that provides S_1 and S_2, S_3 can be deployed with an additional need for resources only for the sake of redundancy. All of its basic functionalities are already provided by S_1 and S_2.

The simple example in (3) shows the advantages of service-agnostic components: While the minimum amount of instances of any context-sensitive component s_i necessary to provide all services in the network equals the number of S_j needing it, the minimum amount of instances of the service-agnostic component t_i is always one. Thinking of redundancy, if n instances of the service-agnostic component t_i exist, *(n-1)* may fail while leaving all services intact. In case of any context-sensitive component s_{ij} however this just guarantees the availability of S_j. So providing redundancy of service-agnostic components helps every service using them while in the case of context-sensitive components this is only true for single services.

These thoughts lead to the conclusion that complex services that are composed of a higher degree of service-agnostic components can ultimately be more survivable since their redundancy is distributed across the whole service network and the replacement of failing functionality is trivial. It should be encouraged to maximize the percentage of independent, service-agnostic functionality. This approach is also feasible for other non-functional attributes of services than redundancy. However it has to be investigated, to what degree complex services actually can be modeled with service-agnostic components as the benefits for different attributes such as availability may vary [JN56].

4 Example

Thinking of an IP-based, service-oriented network this model can be applied to various service classes, all of them being propagated and discovered with zeroconf techniques. Think of a service that reads the temperature from sensors in a room and - after analyzing the data - regulates the heating in that room. The service could be composed of four subtypes of services: temperature sensors, heating regulators, calculation of the adjustment and presentation of the frontend. While obviously the temperature sensors and heating regulators are context-sensitive regarding their location, the calculations can be carried out anywhere where there is enough computation power and also the frontend reading the sensors and setting the controls can be served by any service instance providing the presentation service. So if the frontend realizes that the last used calculation service is gone it can check the zeroconf network for other instances of that service and - if available – switch to using another one.

5 Conclusion

The proposed model provides a concept for survivable architectures. Although this paper focuses on redundancy, the fine granularity of the service description and distribution of their management among simpler, ideally service-agnostic nodes across the whole service network, is believed to improve other non-functional parameters of a complex service as well. The approach promises to better handle run-time diagnostics and on-the-fly (re-)composition of service functionality in networks with highly dynamic capabilities. Furthermore, the variability of the service description provides more possibilities for composing service functionalities. If a vital part of the service breaks down, it might – depending on the application - be recomposed with reduced functionality thus facilitating graceful degradation which is in most cases more desirable than no service functionality at all. How the proposed approach handles those topics needs to be examined in further studies.

References

[DK08] Andreas Dittrich and Jon Kowal. Architektur für selbstkonfigurierende Dienste auf Basis stark ressourcenbeschränkter eingebetteter Systeme. Diploma Thesis, Institut für Informatik, Humboldt-Universität zu Berlin, July 2008.

[Wil02] Aidan Williams. Requirements for Automatic Configuration of IP Hosts. Draft, September 2002. draft-ietf-zeroconf-reqts-12.

[CS05] Stuart Cheshire and Daniel H. Steinberg. Zero Configuration Networking – The Definitive Guide. O'Reilly Media, Inc., 1. Edition, December 2005.

[MM04] Nikola Milanovic and Miroslaw Malek. Current Solutions for Web Service Composition. IEEE Internet Computing 08/2004, pages 51-59.

[JN56] John von Neumann. Probabilistic logics and synthesis of reliable organisms from unreliable components. In C. Shannon and J. McCarthy (editors) *Automata Studies*, pages 43--98, Princeton University Press. 1956.