A Service Oriented Architecture Complexity Metric, Based on Statistical Hypothesis Testing

Michael J. Maynard    George Dimitoglou
Department of Computer Science
Hood College
Frederick, MD 21701

Abstract—Service Oriented Architectures (SOA) is a widely used paradigm for the development and deployment of scalable, loosely coupled distributed environments. Along with the flexibility and relative ease of deployment, such architectures have inherent complexities. These complexities impact the maintainability and testability of SOA implementations but no metrics exist to identify and express these complexities. This paper presents a first attempt towards developing a preliminary complexity metric, the balance factor, which is derived via statistical hypothesis testing and indicates if computing resources are uniformly distributed throughout a SOA-based environment.

Keywords: software architecture, web services, metrics, complexity

I. INTRODUCTION

Centralized system architectures tend to establish static, specific communication and data interchange channels between computing resources. This approach works well in small, static environments where tracking messages interfaces and computations are simpler. Centralization provides better control and harmonization of interactions. However, centralized architectures don’t scale. Distributed environments scale better but their difficulty to manage, control and maintain increase proportionally to the number and heterogeneity of system, components and platforms.

Service Oriented Architectures (SOAs) are inherently distributed and scalable, yet they are amenable to platform, programming language, data type and processing paradigm heterogeneity. The SOA approach is a relatively new software architecture paradigm to deploy, organize and control distributed resources. SOA has already gained widespread acceptance. It is widely used and industry trend studies predict an even more dominant presence for the future [2]. The core component of a SOA—the services (pieces of self-contained access to business logic functionality)—can be designed, developed and deployed by multiple groups, organizations and providers. Technically a service is an interface for various messages so it is viewed as an abstraction that conceals implementation details from the consumers [9]. Another component is the enterprise service bus (ESB), an underlying infrastructure that provides point-to-point, mediation and proxy-based connection interoperability between services and those that consume them.

The integration of these services and an ESB at a high level is trivial, allowing the loose coupling of resources in a distributed system architecture that evolves, often organically, with little or no overall control over size, scope and boundaries. As a result, SOA implementations may be described as a single concrete architecture but determining their boundaries and complexity is arduous, if not impossible. Not knowing the boundaries of a distributed environment obstructs the ability to identify bottlenecks and determine if the SOA is balanced in terms of resources and their consumption. This has direct implications with the manageability, maintainability and cost of the environment.

In software engineering, metrics exist to determine software size and complexity such as
source lines of code (SLOC), the cyclomatic complexity [12] and Halstead measurements. In networking, graph theory provides the mathematical foundations to assess the complexities of network topologies. At a more abstract level in algorithm analysis, metrics do exist to measure algorithm complexity based on asymptotics. Unfortunately, in SOA, no such metrics exist.

The main objective of this work is to introduce and develop a simple model to quantify the notion of balance in a SOA, aiming for metrics similar to the notions of soundness and confidence used in software engineering [3]. A more ambitious objective is to bring to light aspects of the SOA paradigm which will allow other quantitative measurements and ultimately, to be able to establish a single metric that would describe overall architectural complexity.

II. METHODOLOGY

A. Assumptions

To examine and analyze the notion of balance in a SOA certain assumptions are made about the typical SOA architecture. To ensure the generality and pervasiveness of the model we use the fundamental SOA as described by Krafzig et al [8]. In Figure 1, a sample of the fundamental SOA is presented. The major components of the architecture are identified either under the enterprise or the basic layer. The enterprise layer exposes the services to the enterprise and provides the front-end interfaces to service consumers. The basic layer contains all the components required to perform the required computations for the services to be rendered. This layer can be logically divided in two parts, each part from either side of the Enterprise Service Bus (ESB). The ESB is a software architecture construct that acts as a infrastructure conduit between services and the components that implement them. Services may be discrete, such as a simple calculation or a composition of services. In either arrangement the services communicate via the ESB with a collection of back-end mechanisms and repositories.

The mechanisms can also be divided on basic logic services and basic data services. The former perform computations, while the latter facilitate interaction with data repositories and data retrievals.

![Figure 1: The fundamental SOA](image)

Clearly, this fundamental SOA is a skeleton architecture containing only basic data and logic. For the analysis, the focus is on rudimentary, task-completing services. Therefore, a number of relaxing assumptions are made. Data services allow for read and write data operations while logic services provide computation and results.

Composite services are included for completeness but any cost for orchestration is obscured as being part of the ESB functionality. All services are considered “internal” to the SOA. Consequently full disclosure exists about both their exposed and non-exposed functionality.

Issues related to security, reliability and data mapping are excluded. Similarly, higher level issues such as business processes, policies and governance of the architecture are excluded as well.

B. Generalization

Consider any two implementation instances of the fundamental SOA. They could each have a similar profile as the instance illustrated in Figure 2. Both instances would be architecturally identical, offering a set of services, operations and messages, but the implementations and services of each instance would be different. The implemented services could vary, serving as interfaces for different messages and computations, causing the two implementations to vary significantly. This is to be expected, as the notion of service is an abstraction that conceals implementation details from the consumers.
Therefore, a general model of this implementation instance can be derived using web services and operations as the mechanisms to deliver the basic logic and data services. From Figure 3, we can determine initially that a SOA is made up of one or more web services (WS), each of which is comprised of zero or more web service operations (O) along with messages (M) being exchanged between services. Therefore,

$$\text{SOA}_1 = \{\text{WS}_A, \text{WS}_B, \ldots, \text{WS}_N\}$$ (1)

Relation 1, states that SOA$_1$ consists of the set of N web services. A web service is similar to an API or an interface to some back-end computations or capabilities.

A web service is composed of both exposed and non-exposed operations. Web services can be traditional Simple Object Access Protocol (SOAP)-based services or they can also be non-SOAP based such as a Remote Procedure Call (RPC) invocation or a stand-alone module invocation. SOAP based web service operations can be defined, exposed and described using a web service description language (WSDL) file.

This definition describes the characteristics of the service operation without any reference made to the underlying technology which hosts or enables the service to exchange messages [5].

Non-exposed operations are operations that are unavailable as an interface to a client; however the web service uses these operations for internal purposes, similar to a program method within the objected oriented programming language paradigm. Further, it is assumed that an individual SOAP-based web service yields that

$$\text{WS}_A = \{O_1, O_2, \ldots, O_N\}$$ (2)

thus describing each individual web services in terms of the N operations they consists of. For non SOAP-based web services, rather than having distinct operations as they have been already described, they are often one-way invocations of back-end functionality.

The last factor to consider is the messages being transmitted through the SOA. The WSDL file describes the format and details of the messages that are being passed and received. The details include end points, ports, and the message syntax/format.

Additionally, for non SOAP-based web services the messages are often direct exchanges of parameters and data, for example, parameters to a servlet that are appended to the end of a URL.

C. Approach

The fundamental SOA, the generalized services model, and the working assumptions described earlier provide a constrained environment that can be analyzed to determine the balance of a SOA.

The focus of the approach is to dissect the web services of a SOA and attempt to quantify, analyze and attempt to identify meaningful patterns that can provide an indication of the degree of resource utilization and efficiency.

The analysis is based on a simple SOA with five web services working together to geo-rectify an image for a Geography Server (Figure 4).
The operations in this example are based on a workflow task sequence, beginning with the services consumer invoking the Image Service by providing an input image. The Image Service determines the latitude and longitude coordinates associated with the image corner points, via the Latitude and Longitude Services.

The coordinates are passed to a Tile Retrieval Service that will determine, correlate and retrieve all those tiles in the Geography Server associated with the image. The tiles are passed to a Map Overlay Service which makes the association of the image with the tiles and stores this input in the Geography Server.

The geo-rectify example shows the different interactions between web services in a workflow-like fashion to accomplish a specific task. It includes single invocations of web service operations and composition of web service operations to invoke another web service (e.g. the Latitude Service result must be combined with the Longitude Service result to invoke the tile retrieval service). It also demonstrates the interaction with back-end applications (databases) for data logic service calls.

It is apparent from this illustration that certain functions and their associated operations can be assigned to specific SOA components (levels). In Figure 5, the services provider side of the SOA is divided in three levels.

From the perspective of the services consumer, Level 1 is logically closer to the enterprise layer and provides service descriptions, entry points and APIs of exposed services. Level 2 provides the basic data and logic services and Level 3, the logically most distant to the enterprise layer, includes the back-end components such as a data repositories, database management systems (DBMS) or other functionality.

In detail, Level 1 exposes services via WSDL files, which act as an API and provide consumers with the means to interact with the SOA.

Level 2 contains the source code for the operations within each exposed and non-exposed web service. For non-SOAP web services such as RPC invocations, Level 2 may be bypassed and the service works using a direct connection to the back-end layer (Level 3).

Finally, Level 3 includes all the back-end services, applications, and custom code that accomplish the “work” that the SOA is designed for.

Level 3 is the best understood part of the SOA since it contains program code, which can be evaluated by existing methods for developing metrics. Using McCabe’s cyclomatic complexity [12] it is simple to derive many complexity metrics for all the back-end software and provide:

(a) An overall understanding of the flexibility inherent within the organizational structure of the program code, and

(b) The impact and level of difficulty re-factoring the code when changes are introduced.

These particular metrics can offer insight with respect to risk analysis, adaptability, test planning, and re-engineering [3]. Risk analysis provides the means to assess inherent risk during the development cycle. Adaptability measures how well the software may adapt to change over time,
Table 1 - Sample SOA Average Operations

<table>
<thead>
<tr>
<th></th>
<th>WS 1</th>
<th>WS 2</th>
<th>WS 3</th>
<th>WS 4</th>
<th>WS 5</th>
<th>Total Operations</th>
<th>Average Operations per web service</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOA1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>SOA2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>SOA3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>SOA4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>SOA5</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>SOA6</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>SOA7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>

and therefore understanding the expected required level of effort each time a portion of the code is modified or extended. The test planning aspect refers to the ability to determine the number of unique test cases to be generated in order to provide a satisfactory testing state. Level 2 includes the web service operations for SOAP-based web services, along with their code implementations. This level provides insight on how the operations interact with each other, what services call other services, and what services invoke back-end code. Along with the revelation of the SOA inner workings and the flow between services, this is the level that service compositions may be uncovered and described.

Level 3 containing the WSDL files is the most promising level to determine a “balance” factor associated (β) with the SOA. This level provides valuable information such as the number of SOAP-based web services and exposed web service operations, while it also allows for the calculation of the average number of web service operations per web service for the SOA (Table 1).

In Table 1 there are seven different SOAs (SOA1- SOA7), each consisting of five web services. The options differ based on the number of operations per web service and this is reflected in the results of the rightmost columns (total and average number of operations). In this particular comparison, the total number of operations is ten and the average is two across all examined SOAs.

D. Statistical Hypothesis Testing

The presentation of the data in Table 1 revealed the parameters that would affect the balance factor in a SOA. For example, SOA7 is completely balanced, having two operations per web service. The opposite of this is SOA1 where one web service has the majority of operations, and the rest of the services have very few operations. This depicts a SOA that is greatly unbalanced. One operation per service is considered to be minimal since a web service with no operations indicates a web service without functionality.

This distribution of operations per service can significantly skew the results when trying to identify if a SOA is balanced. SOA1 and SOA7 in Table 1 illustrate this example. Both of these SOAs have the same number of total operations and the same average number of operations per web service. Clearly, SOA1 is “unbalanced” with the majority (60%) of all operations concentrated on one service (WS5). At the same time, SOA7 is well-balanced with the number of operations being evenly distributed across all the web services. In the particular example the notion of balance is easily identified due to the small number of operations and web services. However, this may not be the case with larger environments that may contain hundreds of operations and services.
### Table 2 - Sample SOA options with Balance Factor

<table>
<thead>
<tr>
<th></th>
<th>Number of Operations per Service</th>
<th>Total Operations</th>
<th>Average Operations ((\bar{x})) (per web service)</th>
<th>Chi-Squared ((x^2))</th>
<th>Balance Factor ((\beta))</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS 1</td>
<td>1 1 1 1 6</td>
<td>10</td>
<td>2</td>
<td>0.040</td>
<td>4.042</td>
</tr>
<tr>
<td>WS 2</td>
<td>1 1 1 2 5</td>
<td>10</td>
<td>2</td>
<td>0.199</td>
<td>19.914</td>
</tr>
<tr>
<td>WS 3</td>
<td>1 1 1 3 4</td>
<td>10</td>
<td>2</td>
<td>0.406</td>
<td>40.600</td>
</tr>
<tr>
<td>WS 4</td>
<td>1 1 2 2 4</td>
<td>10</td>
<td>2</td>
<td>0.557</td>
<td>55.782</td>
</tr>
<tr>
<td>WS 5</td>
<td>1 1 2 3 3</td>
<td>10</td>
<td>2</td>
<td>0.735</td>
<td>73.575</td>
</tr>
<tr>
<td>WS 6</td>
<td>1 2 2 2 3</td>
<td>10</td>
<td>2</td>
<td>0.909</td>
<td>90.979</td>
</tr>
<tr>
<td>WS 7</td>
<td>2 2 2 2 2</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

By using the chi-squared “goodness of fit” statistical technique, a numeric value can be computed that represents the balance of the SOA. This numeric value summarizes the discrepancy between expected and observed values and is expressed in Equation 1:

\[
\sum_{i=1}^{k} \frac{(x_i - E_i)^2}{E_i}
\]

Equation 1: Chi-squared test statistic

where \(x_i\) is an observed frequency, \(E_i\) is an expected (theoretical) frequency and \(k\) is the number of possible outcomes of each event.

For instance, utilizing the chi-squared technique [4] on the web services for SOA1 against the perfectly balanced web services within SOA7, generates a chi-square \((x^2)\) distribution value. Multiplying this distribution value by 100, results in a “balance” factor as a percentage.

The closer the balance factor \((\beta)\) value is to 100, the more balanced the SOA is in terms of the number of operations spread amongst web services and the distribution and use of resources. Conversely, the closer this number is to zero, the more unbalanced the SOA. Table 2 includes the results of the balance analysis. From the balance factor results, it is apparent that SOA1 is the most unbalanced (balance factor 4.04). The same factor for SOA1 is fairly close to zero, showing that the operations within this SOA are not distributed among the services. This also indicates that a small number of web services are performing the majority of the work hence increasing the possibility of complexity within those unbalanced parts. SOA2 is the most balanced (balance factor 100) which is also obvious by observing the distribution of operations. While in reality, attaining such an ideal balance factor in heterogeneous, distributed environments is not trivial, still, it provides an insight as to the ideal SOA distribution of operations and web services. Consequently, if a deployed SOA could approach the “optimal” balance factor, then this even distribution of operations over services could signify a less complex overall environment. In practice, SOAs tend to resemble the configuration profile of SOA4 and SOA3 (Table 2). Such SOAs are not completely unbalanced, however they contain a few services that handle the brunt of the business logic, and many other services that are used less often and provide very specific and limited operations.

The rationale behind this metric is based on performing a statistical hypothesis test. The broad approach is founded on a method of making statistical decisions from observations and experimental data. This technique is typically used to determine if experimental results contain enough information to indicate high fluctuations in the observed data distributions. In the case of SOA, the higher the fluctuations the more “uneven” and unbalanced the environment appears to be.

Adopting these tests to the evaluation of the SOA, it is seen that calculating the value of average operations per service for almost any environment is possible. This average value \((\bar{x})\) becomes the expected value \((E_i)\) while observed values are the actual numbers of operations per service. If the calculated test statistic is large, then the observed
and expected values are not close and this indicates a poor fit of the observed data against the expected value which in SOA terms indicates an unbalanced environment.

III. CONCLUSION

Considerable work has been done in understanding and representing software complexity. Complexity concepts that can be expressed by specific metrics provide better software understandability, modifiability, testability, maintenance and lead to more efficient software development.

Unfortunately, few such metrics exist in the area of software enterprise systems and particularly in service-oriented architectures. These loosely coupled distributed environments can scale and grow uncontrollably, resulting in environments that are difficult to modify, maintain and test. Being able to measure the degree of complexity within these large environments will help analyze bottlenecks and areas needed for improvement, as well as predict issues that may arise on future growth.

The focus of our work was to devise a metric that would be able to indicate, just by parsing WSDL files from a SOA, those areas in the architecture that are heavily used. The implication from the existence of this metric would be to identify resource-intensive web services and potential bottlenecks that may have direct impact on the maintainability, testability and performance of the SOA.

Devising such metrics required the use of a simplified, yet representative model of a SOA. The fundamental SOA that includes all the elements: an ESB, basic data and logic services and back-end facilities, was used to develop a generalized example that could exercise and test the validity of the devised metric. The result of this study was to determine a metric, the balance factor, indicating how skewed or evenly distributed the resources in a SOA were, regardless of the number of services and operations involved.

This metric is just a first attempt to better understand the notion of complexity in SOAs. Developing a single but comprehensive SOA complexity metric is the goal of future work, which would be similar to the McCabe cyclomatic complexity number for source code.

ACKNOWLEDGMENT

Special thanks to Stephen McCabe of McCabe & Associates and Jim Devilbiss for their insightful comments and discussions.

REFERENCES