PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE

ESCUELA DE INGENIERIA

ANALYZING THE REST ARCHITECTURAL STYLE WITH EXTENDED INFLUENCE DIAGRAMS

FEDERICO M. FERNANDEZ

Thesis submitted to the Office of Research and Graduate Studies in partial fulfillment of the requirements for the Degree of Master of Science in Engineering.

Advisor:

JAIME NAVON C.

Santiago de Chile, (July, 2009)

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERIA

ANALYZING THE REST ARCHITECTURAL STYLE WITH EXTENDED INFLUENCE DIAGRAMS

FEDERICO MARTÍN FERNANDEZ BREYTER

Members of the Committee:

JAIME NAVÓN

ROSA ALARCÓN

LUIS GUERRERO

RODRIGO GARRIDO

Thesis submitted to the Office of Research and Graduate Studies in partial fulfillment of the requirements for the Degree of Master of Science in Engineering.

Santiago de Chile, July, 2009.

This work is dedicated to my parents for their guidance, their support and their unconditional love during all my life.

AKNOWLEDGMENTS

I want to thank my family for all their encouragement and support during the development of this work. Special thanks to my brother Alejandro Fernandez who listened to my presentations and read my drafts several times, helping me a lot with his questions and comments. To my girlfriend Constanza Liborio who I love so much, even thought her beauty delayed the end of this investigation for quite a while. I also want to thank my business partner and friend Anibal San Martin, for his patience and support. I wish to thank my advisor, Dr. Jaime Navon, for allowing me to pursue the research path of my choice, and for being there every time I needed to discuss the smallest thing. I also want to thank Dr. Ben Liblit from UW- Madison for his great support and encouragement during my short stay in that institution. I thank the members of my thesis committee Dr. Rosa Alarcon, Dr. Luis Guerrero, and Dr. Rodrigo Garrido, for kindly accepting being part of this work during the final phase. Special thanks to Dr. Rosa Alarcon, who performed a deep review of this thesis and provided several interesting insights. Also, thanks to all the professors and friends I met during my studies. I want to thank Pontificia Universidad Catolica de Chile for being such a wonderful place to study. Finally, I want to thank the association of former students of my secondary school: Instituto Nacional for the financial aid they gave me during my first year of college. Without that help, I couldn't have studied in this prestigious institution, and this thesis would have never existed.

GENERAL INDEX

TABLE INDEX

FIGURE INDEX

RESUMEN

En este trabajo identificamos y enfrentamos las razones por las cuales la tesis doctoral del Sr. Roy Fielding ha causado tanta confusión entre practicantes e investigadores de REST. De modo de apoyar el análisis de este estilo arquitectónico para su uso en otros dominios, se hace necesaria la existencia de un modelo práctico de la lógica de su diseño. A través de un análisis de los modelos de REST disponibles y usando conceptos traídos de teoría de modelos, describimos los requerimientos de un modelos práctico de REST. Para satisfacer estos requerimientos, proponemos el uso de una combinación de decisiones arquitectónicas y diagramas de influencia extendidos, para generar una representación gráfica trazando la influencia de cada restricción que conforma REST sobre el conjunto de propiedades de software deseadas. El modelo resultante facilita el entendimiento de REST y sirve como un framework para evaluar el impacto de relajar o agregarle más restricciones. Evaluamos el artefacto resultante mediante la demostración de cómo sería utilizado para responder a una difundida preocupación sobre la forma correcta de implementar REST para el desarrollo de aplicaciones web.

Palabras Claves: REST, SOA, WWW, arquitectura de software, estilos arquitectónicos, diseño de software, aplicaciones basadas en red, diagramas de influencia, hipermedia, arquitectura orientada a servicios, computación orientada a servicios, orientación a servicios, orientación a recursos, arquitectura orientada a recursos, mashups, enterprise mashups.

ABSTRACT

In this work we identify and tackle the issues about the doctoral dissertation of Dr. Roy Fielding that has caused so much confusion among practitioners and researchers of REST. In order to support the analysis of this architectural style for its use in other applications domains, a practical model of its rationale is required. Through an analysis of the available models of REST using concepts from modeling theory, we describe the requirements of a new practical model of REST. To satisfy these requirements, we propose to use a combination of architectural decisions and extended influence diagrams to generate a graphical representation tracing the influence of every REST constraint on the set of desired software qualities. The resulting model facilitates the understanding of REST and serves as a framework to assess the impact of relaxing or adding more constraints to it. We evaluate the resulting artifact by showing how it would be used to respond to one widespread concern about the right way to implement REST for web application development.

Keywords: REST, SOA, WWW, software architecture, architectural styles, software design, network based applications, influence diagrams, representational state transfer, hypermedia, service oriented architecture, service oriented computing, service orientation, resource orientation, resource oriented architecture, mashups, enterprise mashups.

1 INTRODUCTION

1.1 Problem Context

Due to the great success of the WWW architecture not only in scalability, but also in simplicity and flexibility, there is increasing interest in understanding and applying the principles behind the WWW architecture to solve problems of other domains. There isn't consensus in the field of Computer Science about the concept of Software Architecture, but it is generally accepted that the WWW architecture is primarily characterized by the standards: HTTP, URI, and media types (like HTML, JPG and XML).

In 2000, one of the authors of the HTTP 1.1 and URI specifications released his Ph.D. dissertation under the name of "Architectural Styles and the Design of Network Based Architectures" (Fielding, 2000). In the 5th chapter of that document the author explains the design rationale of REST, an architectural style that would explain the principles behind the WWW architecture. Since that work, many researchers and software developers have discussed possible applications of REST for other problem domains than that of the original WWW.

REST and the WWW protocols have been specially considered as an alternative to SOAP, WSDL, UDDI and the rest of the WS-* specifications for application to application communication. In practice, this means integrating applications using HTTP at the application layer, instead of using it as the transport for other protocols. This approach had great success among web application developers ("API Dashboard: ProgrammableWeb," 2009) who started to use tools to provide access to their applications through an interface composed of a set of URIs, the HTTP verbs that can be applied to those URIs, and additional information like the

parameters required by each request or the representations available for each resource. Such kind of interface was given the name of “RESTful API” or “RESTful Web Service”.

The success of the RESTful APIs can be observed in the proliferation of Mashups, which are applications produced by combining remote functionality that is available through RESTful APIs. These applications motivated web application developers to maximize the simplicity of opening their web applications for external consumption, by making them as compliant as possible with the HTTP semantics during the design phase. One set of guidelines for designing applications in that way was provided by (Richardson & Ruby, 2007). They called it “Resource Oriented Architecture” and defined it as “a way of turning a problem into a RESTful web service: an arrangement of URIs, HTTP, and XML that works like the rest of the Web, and that programmers will enjoy using”.

The success of the RESTful APIs in the WWW also motivated IT consultants and researchers coming from the Service Oriented Architecture context. The idea of Enterprise Mashups was introduced (Altinel et al., 2007; Floyd, Jones, Rathi, & Twidale, 2007; Jhingran, 2006; Wong & Hong, 2007) as a way to facilitate the construction of quick, situational applications in the enterprise. In addition, REST principles were introduced as an alternative to the traditional ways in which Service Oriented Architecture is implemented in the enterprise.

However, as REST and the WWW architecture are not the same thing, but, at the same time, the concepts of software architecture used in (Fielding, 2000) are not universally known nor accepted, there was a lot of confusion about what REST is, and what it should be compared to. From time to time new publications were released clarifying REST concepts (S. Vinoski, 2008a, 2008b; Steve Vinoski, 2007) but over time it became apparent that the only person who really understood

REST was his author, who refused several widespread practices among practitioners of REST (Fielding, 2009, 2008b). Even the academic publications that tried to clarify the debate (Overdick, 2007; Pautasso, Zimmermann, & Leymann, 2008) couldn't survive to the clarifications made in the posts cited above. In order to avoid this problem, some IT consultant started to dismiss REST and focus instead on just reusing for SOA the technologies that have worked in the WWW, like HTTP, cookies, Javascript, and the proven solutions that have been built using them. They called this approach "Web Oriented Architecture" (Hinchcliffe, 2008). Other IT consultants started new discussions for a better understanding of REST, and its practical application for SOA (Allamaraju, 2008).

This is where we stand today regarding the potential application of the WWW architecture and/or the principles that guided its design to other problem domains.

1.2 Motivation

Within the problem context explained above, there are several questions that motivate our research:

a) Why has REST been so hard to understand? Is it because of a lack of effort from the stakeholders, or is it because there are barriers inherent to the dissertation that introduced the term?

b) What is the real meaning of the concepts that surround REST: architectural styles, software architectures, architectural properties? What is the relation between these concepts, and those more generally used in the systems and modeling literature, like models and meta-models?

c) Given that REST is an architectural style designed for the requirements of the standard WWW architecture, why is everybody trying to apply

REST to SOA? Shouldn't we instead try to design a custom architectural style for a standard SOA architecture? What are the barriers to accomplish this task?

1.3 Hypothesis

If the design rationale of the standard WWW architecture could be described with a practical model, a usable artifact, then that artifact could be used to reason about the applicability of the WWW architecture to solve different problems. It could also be used to reason about producing other architectures, similar but different to the successful WWW. Supposing that the design rationale of the standard WWW architecture has been properly documented in (Fielding, 2000) under the name of REST, then a practical model of the design rationale of the WWW architecture could be obtained from the development of a novel representation of the information contained in that work. This representation should facilitate the visualization of REST and the consequences of modifying parts of it. It would serve as a tool to reason about applying the principles behind the successful WWW architecture to other problem domains, and it would help to answer the questions stated in the previous subsection.

1.4 Objectives

The general objective of our research was to contribute to the study the possible applications of the WWW architecture and its design rationale to the realization of Service Oriented Architecture. Because of the hypothesis presented in the previous section, the first step to do that would be to develop a practical model of REST. Therefore, that constitutes the goal of this work. To reach it, we had to perform a deep revision of the doctoral dissertation that introduced REST, discover its flaws;

analyze them using concepts from modeling theory, find a good artifact to represent REST design rationale, and develop the practical model of REST using this artifact. The next section explains this process and the result of our research.

2 PAPER

2.1 Introduction

Due to the great success of the WWW architecture not only in scalability, but also in simplicity and flexibility, there is increasing interest in understanding and applying the principles behind the WWW architecture to solve problems of other domains. In particular, several debates have emerged in the past few years related to the applicability of REST to build better web applications (Richardson & Ruby, 2007) and as an alternative to Web Services for the realization of SOA in the enterprise (Pautasso et al., 2008).

The debates have been diverse in the sense that the actual meaning of REST is different in many of them. For example, we have seen debates about REST vs SOAP (Muehlen, Nickerson, & Swenson, 2004), REST vs Web Services (Pautasso et al., 2008; Xu, Zhu, Liu, & Staples, 2008), and Resource Orientation vs. Service Orientation (Overdick, 2007). Many times researchers have been proved to be wrong about what they think REST is. From time to time, the author of REST clarifies a widespread confusion about the term (Fielding, 2009, 2008a, 2008b), and a new wave of publications appears about how to apply it to different problem domains. Unfortunately, this generalized misunderstanding of the term has produced religious debates between supporters of REST and the supporters of other technologies/principles.

But why is REST so hard to understand? We believe that the reason is that the doctoral dissertation that introduced the term was not particularly written for the readers looking for reusing REST or parts of it for new application domains. Therefore, it is not as clear as it could be for this purpose and, more importantly, it does not provide the conceptual models required by this kind of readers.

In this work we review the dissertation that introduced REST, identify some inherent issues that prevents its correct understanding, classify the available models of REST using concepts from modeling theory, and propose a new practical model that is suitable to perform studies about its possible application to the solution of different problems than those faced by the standard WWW architecture. The model was populated with data taken through a deep scan of the dissertation to make sure that it covers everything that was included in it and does not add anything else.

The outline of this article is as follows. We start in section 2 with a review of related work. In section 3 we summarize the doctoral dissertation that introduced REST and identify the issues that cause confusion among practitioners. In section 4, we analyze REST background material using a set of fundamental concepts from modeling theory. In section 5, we use these concepts to describe the current models of the structure and design rationale of REST and to explain the changes that are required to make them practical. In section 6, we introduce our proposed practical model of REST. In section 7, we show how we filled this model by extracting the knowledge available in the dissertation, and present some insights that can be inferred immediately from the resulting artifact. In section 7 we evaluate the applicability of our model by using it to respond to a question concerning practitioners of REST in the context of web application development. Finally, in section 8 we present our conclusions.

2.2 REST Dissertation: Summary and Identified Issues

2.2.1 Summary

REST is usually referred to as an architectural style introduced in the Chapter 5 of the Ph.D. dissertation of Dr. Roy Fielding ("Dissertation" and "Author" for the rest of this document). While this is true, in order to understand REST it is necessary to read the full Dissertation. The reason is that REST rationale can't be understood if one misses the background material of Chapters 1-3 and the description of the WWW architecture requirements of Chapter 4.

The following is a brief summary of the contents of each chapter of the Dissertation. In the rest of this article we will reserve the words Chapter 1, Chapter 2, Chapter 3, Chapter 4, Chapter 5, and Chapter 6 to refer to the chapters of the Dissertation.

Chapter 1 introduces the concepts about software architecture required to understand the dissertation: software architecture, architectural elements, architectural styles, architectural properties, and architectural design.

Chapter 2 declares that the scope of the dissertation is the study of network based applications architectures, and describes a hierarchy of architectural properties (software qualities) that will be used to compare different known network based applications architectural styles.

Chapter 3 compares 21 different network based architectural styles considering their impact to the set of architectural properties declared in Chapter 2. The Author decided to use a qualitative approach to describe these impacts. Thus, he described

with text the positive and negative impacts of the styles over the software qualities, and used that information to fill the cells of Table 1 with a plus sign to indicate a positive impact or a minus sign to indicate a negative impact. When a style affected a software quality both positively and negatively (something possible due to the fact that the software qualities listed are relatively coarse grained, for the sake of simplicity and visualization) then both a plus and minus sign were written in the cell. The acronyms present in Table 1 stand for the following architectural styles:

a) PF: Pipe and Filter
b) UPF: Uniform Pipe and Filter
c) RR: Replicated Repository
d) $: Cache
e) CS: Client-Server
f) LS: Layered System
g) LCS: Layered Client-Server
h) CSS: Client-Stateless-Server
i) C$SS: Client-Cache-Stateless-Server
j) LC$SS: Layered-Client-Cache-Stateless-Server
k) RS: Remote Session
l) RDA: Remote Data Access
m) VM: Virtual Machine
n) REV: Remote Evaluation
o) COD: Code on Demand
p) LCODC$SS: Layered-Code on Demand-Client-Cache-Stateless-Server
q) MA: Mobile Agent
r) EBI: Event-Based-Integration
s) C2: C2
t) DO: Distributed Objects
u) BDO: Brokered Distributed Objects

The Author describes some architectural styles as derived from other styles. When a style is derived, he shows its predecessors in the column labeled “Derivation”. Unfortunately, not all the styles or constraints that are part of the derivation of a style are shown in the table. Therefore, the impact of a derived style on the set of software qualities (the plus and minus signs in each row) is not always a simple union of the impact of its predecessors. This is the case with the following styles: UPF, $, CSS, RS, RDA, REV, COD, MA, DO, and BDO. As we will see in the next section, it will be important to identify the missing parts in order to be able to reason about relaxing or adding constraints to REST.

One thing we discovered while reviewing this table is that some data is missing: 1) the Uniform Interface style (U) that will be used to derive REST in Chapter 5 is not listed and 2) a couple of plus signs were absent. Table 2 shows a modified version of Table 1, including the plus signs of COD and LCODC$SS that were missing, and two new rows: one for Uniform Interface (U) and one for REST. The marked cells are those whose signs don’t correspond to any possible union of the styles they derive from.

Style	Derivation	Net Perform.	UP Perform.	Efficiency	Scalability	Simplicity	Evolvability	Extensibility	Customiz.	Configur.	Reusability	Visibility	Portability	Reliability
PF			±			+	+	+		+	+			
UPF	PF	−	±			++	+	+		++	++	+		
RR			++		+									+
$	RR		+	+	+	+								
CS					+	+	+							
LS			−		+		+				+		+	
LCS	CS+LS		−		++	+	++				+		+	
CSS	CS	−			++	+	+					+		+
C$SS	CSS+$	−	+	+	++	+	+					+		+
LC$SS	LCS+C$SS	−	±	+	+++	++	++				+	+	+	+
RS	CS			+	−	+	+					−		
RDA	CS			+	−	−						+		−
VM						±		+				−	+	
REV	CS+VM			+	−	±		+	+			−	+	−
COD	CS+VM		+	+	+	±		+		+		−		
LCODC$SS	LC$SS+COD	−	++	++	+4+	+±+	++	+		+	+	±	+	+
MA	REV+COD		+	++		±		++	+	+		−	+	
EBI				+	− −	±	+	+		+	+	−		−
C2	EBI+LCS		−	+		+	++	+		+	++	±	+	±
DO	CS+CS	−		+			+	+		+	+	−		−
BDO	DO+LCS	−	−				++	+		+	++	−	+	

Table 1: Architectural styles vs. software qualities

Style	Derivation	Net. Performance	UP Perform.	Efficiency	Scalability	Simplicity	Evolvability	Extensibility	Customization	Configuration	Reusability	Visibility	Portability	Reliability
PF			+-			+	+	+		+	+			
UPF	PF	-	+-			++	+	+		++	++	+		
RR			++		+									+
$	RR		++	+	+	+								
CS					+	+	+							
LS			-		+		+							
LCS	CS+LS		-		++	+	++				+		+	
CSS	CS	-			++	+	+					+		+
C$SS	CSS+$	-	+	+	++	+	+					+		+
LC$SS	LCS+C$SS	-	+-	+	+++	++	++				+	+	+	+
RS	CS			+	-	+	+					-		
RDA	CS			+	-	-						+		-
VM						+-		+				-	+	
REV	CS+VM			+	-	+-		+	+			-	+	-
COD	CS+VM		+	+	+	+-		+	+	+		-		
LCODC$S	LC$SS+CO	-	++	++	+4+	++-+	++	+	+	+	+	+-	+	+
MA	REV+COD		+	++		+-		++	+	+		-	+	
EBI				+	--	+-	+	+		+	+	-		-
C2	EBI+LCS		-	+		+	++	+		+	++	+-	+	+-
DO	CS+CS	-		+			+	+		+	+	-		-
BDO	DO+LCS	-	-				++	+		+	++	-	+	
U				-		+	+					+		
REST	LC0DC$SS	-	++	++-	+4+	++-++	+++	+	+	+	+	+-+	+	+

Table 2: Architectural styles vs. software qualities, extended

Chapter 4 explains the requirements that of the WWW architecture and the problems faced in designing and evaluating proposed improvements to its key communication protocols.

Chapter 5 introduces REST: the architectural style used to guide the development of the standard protocols that constitute the WWW architecture. The derivation of REST from other architectural styles is presented with the use of a style derivation tree (Figure 1). All the styles used in the derivation are from the list presented in Chapter 3, except for the Uniform Interface style that is characterized as a composition of "four interface constraints": identification of resources, manipulation of resources through representations, self descriptive messages, and hypermedia as the engine of application state.

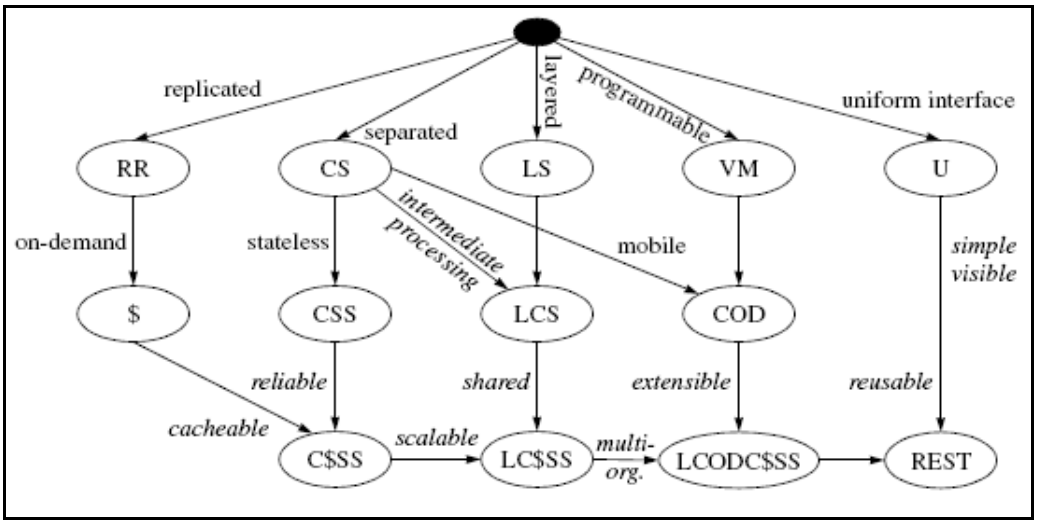

Figure 1: REST derivation tree

Finally, Chapter 6 describes how REST was applied while authoring the internet standards for HTTP and URI, and comments on the mismatches between several Internet technologies and REST.

2.2.2 Issues

Many practitioners and researchers trying to apply REST to other contexts like the one of enterprise information systems have found that the dissertation is hard to understand. The proof for the practitioners is that in the past few years, false beliefs about REST proliferated among web application developers, who even wrote books about "RESTful" development (Richardson & Ruby, 2007) that proved to be wrong in many concepts when the Author decided to create a blog and clarify some misunderstandings about REST (Fielding, 2009, 2008b). The proof for the researchers can be obtained by the fact that different researchers use the term REST to refer to different things.

In part, these misunderstandings are a consequence of the fact that the concepts introduced in Chapter 1 are not universally known nor formally explained**:** the Author declares the definitions he uses of software architecture, architectural styles, and his idea of architectural design. These definitions are not standard, so an important step before attempting to reuse REST is to understand the concepts explained in Chapter 1. For this reason, there would be benefit in re-explaining the concepts of Chapter 1 using more common ground concepts. This is something that we will try to do in section 2.3.

In addition, we argument that the aforementioned misunderstandings are caused by the following issues inherent to the Dissertation:

(a) **The goals driving REST derivation are not explicit:** The derivation process of REST consists of combining styles in order to obtain an architectural style that will induce a set of desired properties that is a superset of the software qualities required by a successful WWW architecture. According to the definitions of Chapter 1, some qualities of the superset will be more important to the WWW architecture than others, and that should drive the selection of styles to include in the derivation of REST. However, the set of desired software qualities of the WWW are not clearly defined. They can only be induced indirectly by relating the qualities of Chapter 2 and the requirements explained in Chapter 4. In addition, the importance of the different qualities presented in Chapter 2 with regards to the WWW architecture is not explained.

(b) **The rationale as to why each style in REST was selected instead of its alternatives is missing**: Some of the architectural styles presented in Chapter 2 are used in Chapter 5 to derive REST. However, no reasons are given as to why some styles are used and other excluded; only a commentary is given for each one of them. For example, it is not explained why the C2 architectural style is not used in REST's derivation.

The reason is not that C2 fails to induce desired software qualities, because as we can see in Table 1 it is one of the styles with most positive impacts. The reader understands that the author considered that this style was not appropriate for the WWW, but the rationale is not explained.

(c) **The uniform interface is not characterized as an architectural style**: Chapter 5 introduces Uniform Interface as a style that is part of the REST derivation tree (Figure 1). This style is presented as the most distinguishing feature of REST, but it is not included in the classification of Chapter 2. Therefore, it is not clear if it is a pure or derived style. Chapter 5 explains it as composed of "four interface constraints": identification of resources, manipulation of resources through representations, self descriptive messages, and hypermedia as the engine of application state. These constraints and their relationships are described in section 5.2 of the dissertation, but the contributions of each one of these constraints to the style are not clear as it is in Chapter 2. We believe that this causes much of the confusion among REST practitioners. How important is each uniform interface constraint to reach REST's induced properties? How important are the styles used and described in Chapter 2 compared to the Uniform Interface presented in Chapter 5?

These issues are all manifestations of one fundamental problem: the Dissertation doesn't provide the conceptual models that are needed to visualize REST rationale, and to reason about potential modifications and applications to other problem domains

Before presenting the model we propose, we will support the assertions above using some concepts from modeling theory.

2.3 Analyzing REST background material

As the concepts explained in Chapter 1 are not universally accepted nor used, it is very important to understand them before studying the application or reusing the principles behind REST to other domains. We start this subsection by citing the most important definitions of Chapter 1:

a) **Software System**: *"... A (software) system may be composed of many levels of abstraction and many phases of operation, each with its own software architecture."*

b) **Software Architecture**: *"A software architecture is an abstraction of the run-time elements of a software system during some phase of its operation. It is defined by a configuration of architectural elements constrained in their relationships in order to achieve a desired set of architectural properties."*

c) **Architectural Elements**: *"The architectural elements of any architecture can be classified in components, connectors and data… A component is an abstract unit of software instructions and internal state that provides a transformation of data via its interface.... A connector is an abstract mechanism that mediates communication, coordination, or cooperation among components… A datum is an element of information that is transferred from a component, or received by a component, via a connector."*

d) **Properties of a Software Architecture:** *"The set of architectural properties of a software architecture includes all properties that derive from the selection and arrangement of components, connectors, and data within the system. Examples include both the functional properties achieved by the system and non-functional properties, often referred to as quality attributes."*

e) **Goal of architectural design**: *"The goal of architectural design is to create an architecture with a set of architectural properties that form a superset of the system requirements. The relative importance of the various architectural properties depends on the nature of the intended system."*

f) **Architectural Styles**: "*An architectural style is a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style"*

g) **Relationship between Architecture and Architectural Styles***: "New architectures can be defined as instances of specific styles. Since architectural styles may address different aspects of software architecture, a given architecture may be composed of multiple styles. Likewise, a hybrid style can be formed by combining multiple basic styles into a single coordinated style."*

In order to better understand the relationship between REST and the WWW, and more importantly, between REST and other concepts like Service Oriented Architecture, we have to map the concepts above to more fundamental concepts of systems modeling theory. We will use the terminology and rationale presented in (Kühne, 2006) because it includes fundamental and unifying definitions of terms like systems, models, and abstractions.

2.3.1 Brief Summary of Küne's Modeling Terminology

According to (Kühne, 2006) there are two fundamental types of models: token models and type models. Token models are models whose abstraction process includes only projection and translation. Type models are different in that their abstraction process also includes classification. A typical example of a token model is a map (Figure 2). A map projects the information of geography into a representation with fewer dimensions, and translates it to a given language (like boxes and lines). A requirement is that the projected data must maintain all the original relationships. In addition, the elements can't be grouped, there has to be a 1 to 1 relationship between the token model elements and the system elements.

Token models are transitive, i.e. the token model of a token model of a system is also a token model of the system.

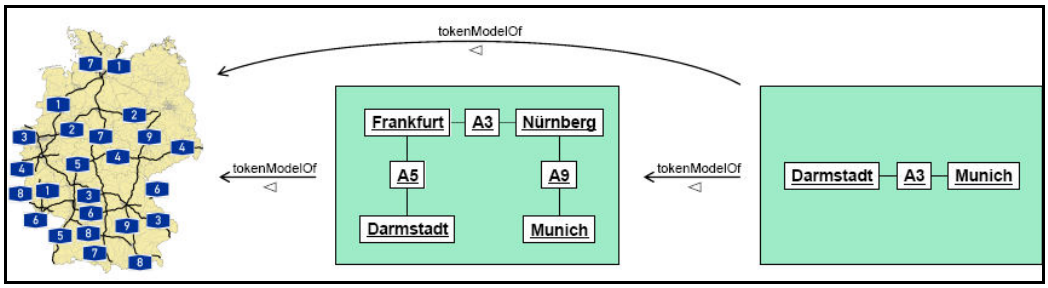

Figure 2: Example of a token model

An example of a type model is presented in Figure 3. Type models are characterized by its ability to classify several system elements into a few model elements. Type models are not transitive; a consequence of this is that a type model of a type model of a system can satisfy the intuitive notion of what we call a meta-model.

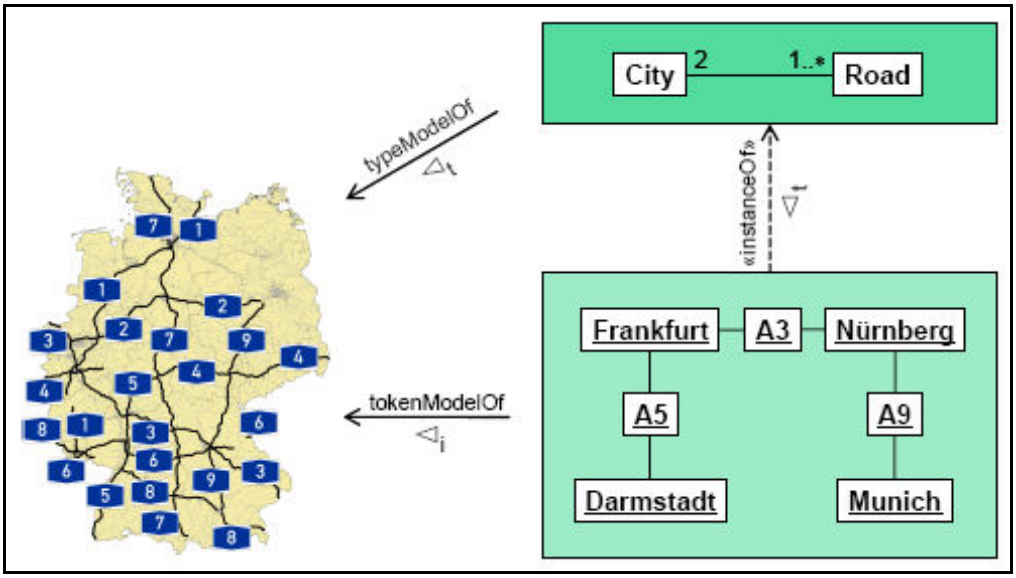

Figure 3: Example of a type model

The last concept required for the rest of this article is “generalization”. We said that the abstraction process leading to a token model includes projection and translation, and the one leading to a type model includes projection, translation and classification. However, there is another possible operator called “generalization”. When in the abstraction process generalization is applied instead of classification, we get a Generalized Type Model. This kind of abstraction process only makes sense if done over existing type models. What is important about this is that the generalization type model of the type model of a system is also a type model of the system. We will see that this is important to understand how to fix the representation of the REST architectural style presented in the Dissertation.

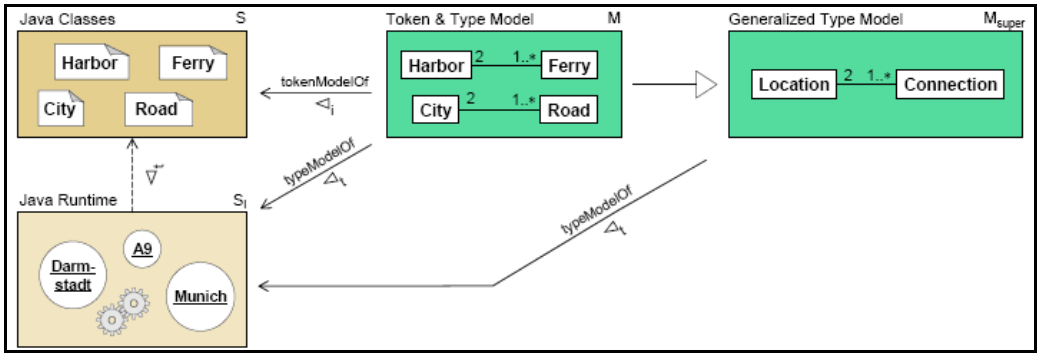

Figure 4: Example of a generalized type model

2.3.2 Architectural Styles in Terms of Token and Type Models

We can describe the concepts of software architecture and architectural styles as token and type models:

a) A software architecture is a token model of the system under study. In practice, this means that a software architecture is a representation of (one part of) the software system that hides a lot of details to make it easier to understand. For example, a description of the components and their interactions of the Amazon

shopping cart software running today would be a software architecture representing the shopping cart part of the Amazon software system running today.

b) An architectural style is a type-model of many software architectures, because it satisfies the notion of classification. Any architecture satisfying the coordinated set of constraints defining a given architectural style is considered to be an instance of that style.

In the next section we will use this conceptual mapping to evaluate the models of REST provided in the Dissertation.

2.4 Analysis of Available REST Models

2.4.1 Models of the REST Design Rationale

In the Dissertation, the Author describes the REST *design rationale* using a REST derivation tree (Figure 1), the matrix of styles vs. architectural properties (Table 1), and text. These can be seen as three token models of the REST design rationale (Figure 5). In order to understand REST design rationale, one has to study the three models, and it is very hard to reason about modifications or applications of the style to other contexts.

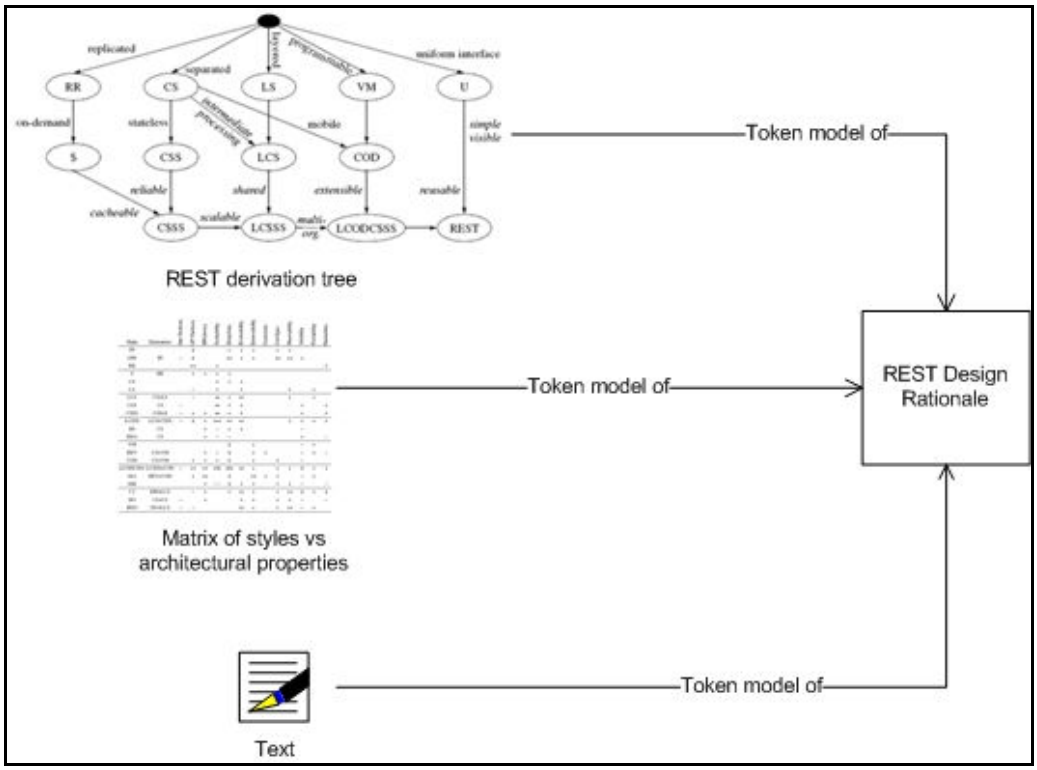

Figure 5: Current models of REST design rationale

What we would like to have is one unified token model of the REST design rationale that we could manipulate to visualize changes in a concise manner. We will describe the requirements of such a model in the next section. As we will see in section 2.7 we will decide to use something called extended influence diagrams.

Figure 6 shows the REST influence diagram in context.

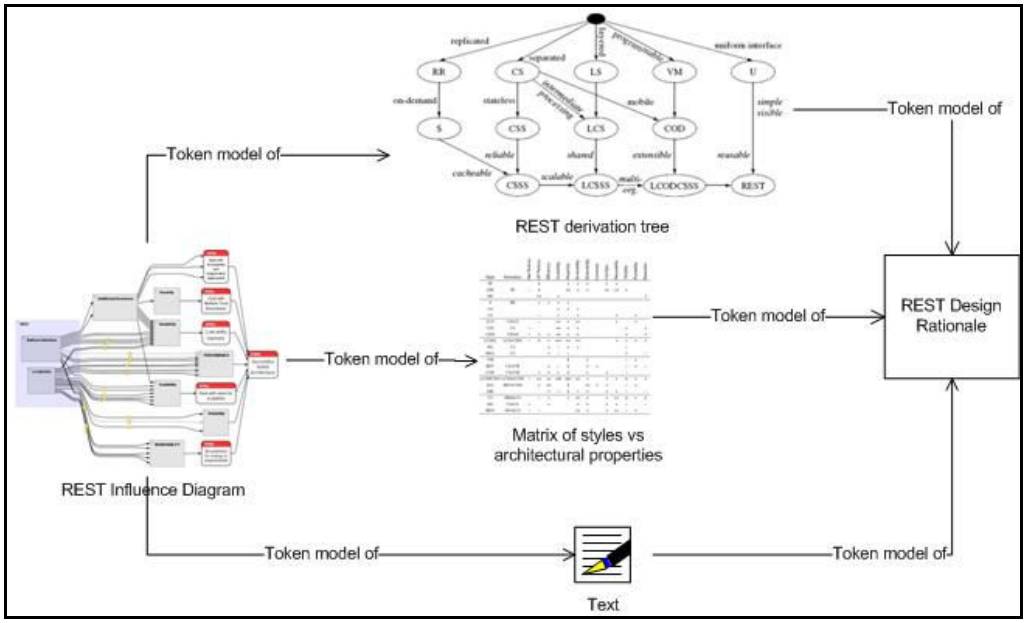

Figure 6: Our proposed model of REST design rationale

2.5 Models of REST Structure

To describe REST (not its design rationale), the Author defines the architectural elements (instances of components, connectors and data) present in REST, and uses a process view to show some possible configurations of these elements (Figure 7). As Figure 8 shows, this is a token model of the set of all the possible configurations of the architectural elements that are defined for REST (client, server, proxy, cache, tunnel, resolver, resource, representation, etc). This kind of model is useful to describe an architectural style, but it is not practical to use as part of a model of its design rationale. What we would like to have is some kind of representation of REST as a set of constraints.

REST, as the architectural style it is, is usually understood as a coordinated set of constraints that has to be respected by any architecture to be called “RESTful”. However, there is no concrete model for this intuitive notion, which is what we

would like to have to visualize the impact of adding or removing constraints to REST.

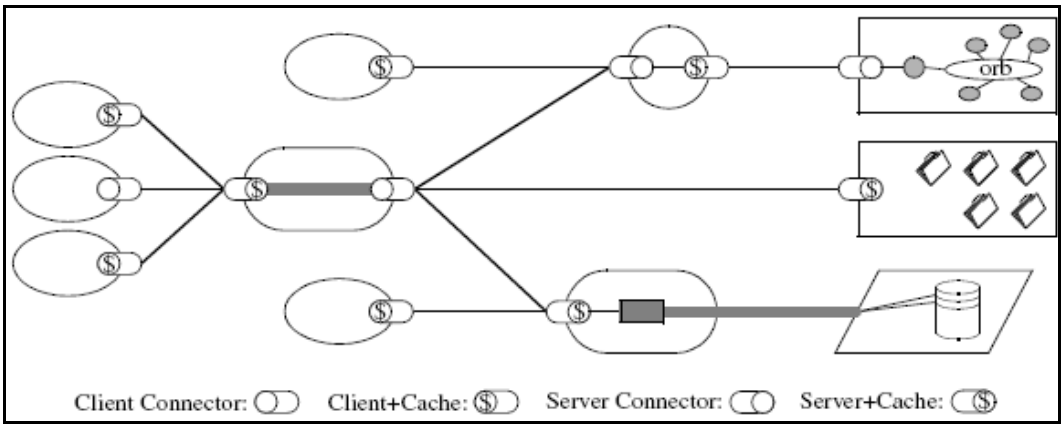

Figure 7: Process view of REST

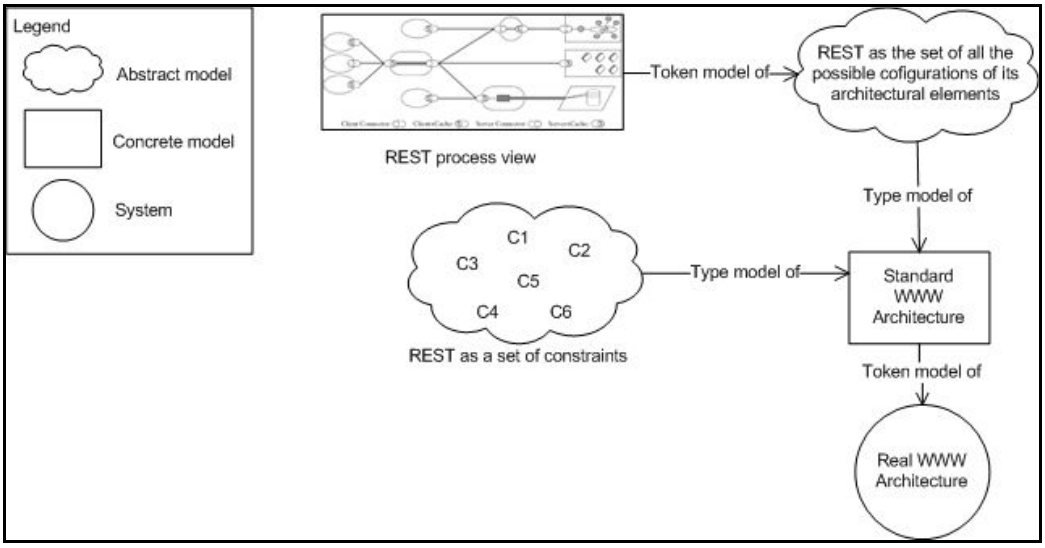

Figure 8: Current models of REST structure

We will propose a new model of REST structure in the next subsection.

2.6 Towards a Practical Model of REST

In this section we explain our proposed models to describe the structure and design rationale of REST in a form that facilitate its understanding and are suitable to reason about its potential reuse for other problem domains.

2.6.1 Defining REST as a sequence of architectural decisions

Inspired by the concept of derived styles introduced in the presentation, we will describe REST as a generalized type model (see section 2.3.1) by using the concept of architectural decision. We will define them in the following manner:

An architectural decision is a named set of constraints that can be added to an architectural style. The result of adding an architectural decision to an architectural style is another architectural style.

A corollary of this definition is that a given architectural decision can only be made over certain architectural styles. For example, we could say that the architectural decision called: “Stateless CS Interactions” only makes sense if applied over the Client-Server architectural style. What we win with the introduction of this concept is that we can now describe an architectural style as a sequence of architectural decisions. Figure 9 shows this model of REST in context.

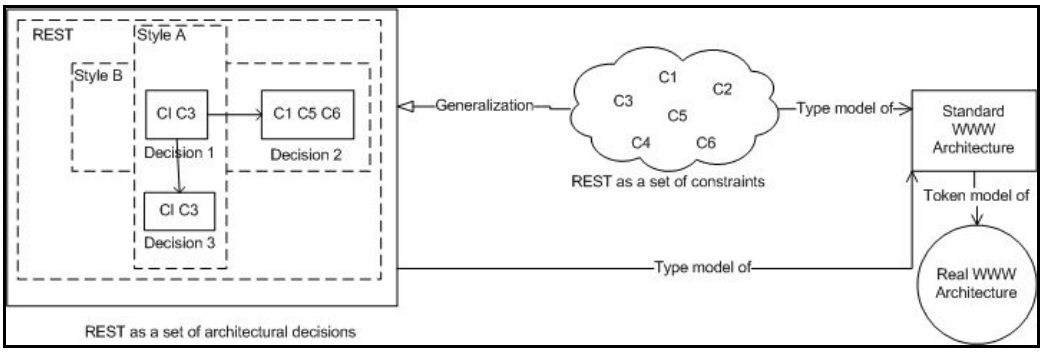

Figure 9: Our proposed model of REST structure

The difference with the "derived style" approach introduced in the Dissertation is that we make every component of an architectural style explicit. Therefore, the properties of an architectural style become the cumulative properties of its individual architectural decisions. This makes the generalized model a great component for the practical model of the REST design rationale.

2.6.2 Requirements of a Practical Model of REST Design Rationale

As it was explained before, we would like to have a unified token model of the REST design rationale that we could manipulate to visualize changes in a concise manner. More specifically, the user should be able to:

a) Visualize and understand how each one of the architectural decisions of REST impacts the set of goals that guided its design. (R1)

b) Visualize and understand the changes caused in the induced properties if new architectural decisions are added to REST, or if existing decisions are replaced for others. (R2)

c) Visualize the set of alternative architectural decisions for each decision in REST. (R3)

d) Easily modify the REST model to visualize and understand how each one of the architectural decisions of REST would impact a different set of goals. (R4)

In addition, there is one non functional requirement that has to be met:

e) The model should be a loyal representation of the Dissertation in order to be accepted by the stakeholders. (R5)

2.6.3 Candidate Solutions

We explored several alternatives before deciding to go with the solution presented in the next section. First, we considered extending the classification framework of Table 2 by adding rows in the top that would group the different properties into broader categories, as a hierarchy of desired properties. This approach would satisfy the visualization part of R1 and R2, and maybe R4, but would not improve understandability, since the user would have to read through the Dissertation to understand the meaning of the plus and minus signs.

Secondly, we considered representing REST as a set of documents, one per architectural decision, containing a description of the decision and the alternatives discarded (Jansen & Bosch, 2005) but we discarded this approach because it wouldn't have satisfied R1, R2 and R4.

Finally, we considered to use graphical models. After some research, we decided to use a subset of the extended influence diagrams defined in (Johnson, Lagerström, Närman, & Simonsson, 2007). For our purposes it was enough to use the utility, chance and decision nodes, and just one type of arrow that would mean different things depending on the type of the connected nodes. As we are not trying to make a model capable of probabilistic reasoning, the resulting graphical notation is very close to the one explained in (Chung, Nixon, Yu, & Mylopoulos, 1999) for software development based on non-functional requirements.

The visibility part of R1 and R2 would be met using collapsible graphical elements. The understandability part would be met by adding as many kinds of boxes as necessary to trace the impact of a decision over the set of goals. R4 would be satisfied by drawing hierarchies of goals from the most general ones, to those represented by software qualities. R5 would be satisfied by extracting all the knowledge to define the elements of the diagram and their relationships from the Dissertation, and documenting all these definitions with comments in the diagram. The only requirement that would not be satisfied is R3, but the ability to trace causality from decisions to goals should help the user identify possible alternatives for each decision.

2.6.4 Summary

In this section we proposed to describe the structure of REST as a sequence of architectural decisions, with each decision implicitly representing a set of constraints over a set of architectural elements. In order to describe the design rationale of REST, we proposed to use extended influence diagrams, which are probabilistic graphical models with three kinds of nodes: decisions, chances and utilities. The architectural decisions that describe the structure of REST are used to define the decision nodes in the extended influence diagram of REST. Therefore, the first proposed model is used in the second one.

In the next section, we will explain how we built a practical model of REST using extended influence diagrams.

2.7 Analyzing REST with Extended Influence Diagrams

In order to make the diagrams, we relied on a flexible visual modeling tool (*Flying Logic*, s.d.). As it also served as a utility to browse, zoom, collapse and expand parts of the produced diagrams, we decided to adapt the syntax of the extended influence diagrams described in (Johnson et al., 2007) to the types of nodes and arrows available in the modeling tool.

Figure 10 shows the resulting syntax. There are three node types and three relationship types.

The node types are:

a) Decision nodes: represent architectural decisions.

b) Utility nodes: represent the desired goals of the target architectural style. They are impacted by each architectural decision. In the case of the REST influence diagram, the utilities are extracted from Chapter 2 and 4 of the Dissertation.

c) Chance nodes: they describe the chains of causal effects starting from the decisions and ending on the utility nodes.

The three relationship types are causal relationship, definitional relationship and temporal relationship. As the modeling tool has only one kind of arrow, the relationship type is defined by the nodes being connected and the color of the arrow:

a) An arrow connecting a decision with a chance means that the decision has a causal effect over the chance. A black arrow means a positive effect, while a grey arrow means a negative effect.

b) An arrow connecting a chance with a utility means that the chance has a causal effect over the utility. Only a black arrow, meaning a possible effect, is allowed.

c) An arrow connecting two chances means that the first chance has a causal effect over the second one. Only a black arrow, meaning a possible effect, is allowed.

d) An arrow connecting two decisions means that one decision was made before the other.

e) An arrow connecting two utilities means that one utility is a child of the other. The parent utility is comprised of its constituent children.

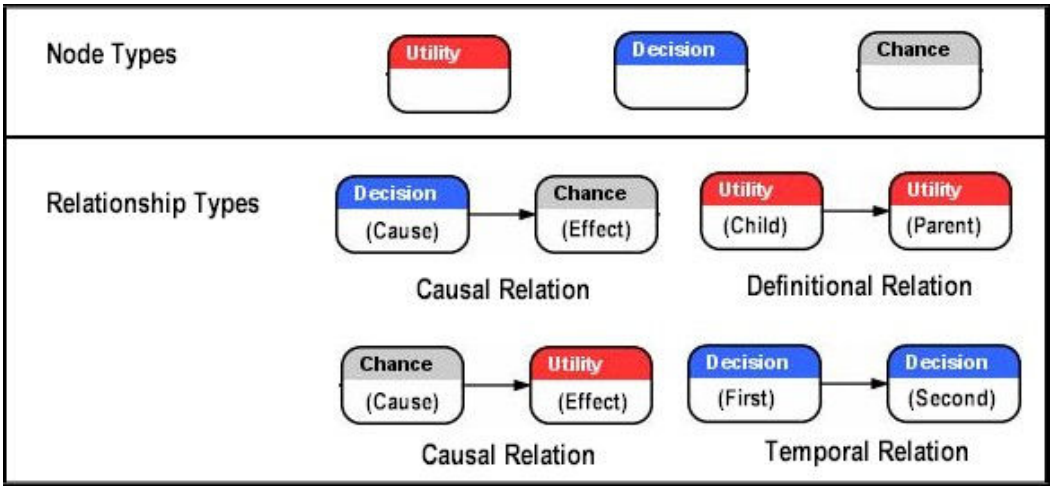

Figure 10: The syntax of the REST influence diagram

In order to improve visibility, we didn't use arrows to connect decision nodes most of the time. Instead, we described the sequence of architectural decisions that defines REST as nested decision sets. This improved visibility at the cost of reduced understandability and expressiveness. The visibility is improved because the arrows going from Decision to chances are easier to draw and follow. Understandability is reduced because if a decision is outside a given set, it means that it was made after the decisions inside the set, but if two or more decisions are

in the same set, the user can't know which was done before the other, and has to get this information from another source (e.g. comments). The expressiveness of the diagram is reduced because only some sequences of decisions can be grouped as architectural styles. Figure 11 shows an example of how a group of temporally related architectural decisions would be represented in our extended influence diagram, using the aforementioned modeling tool.

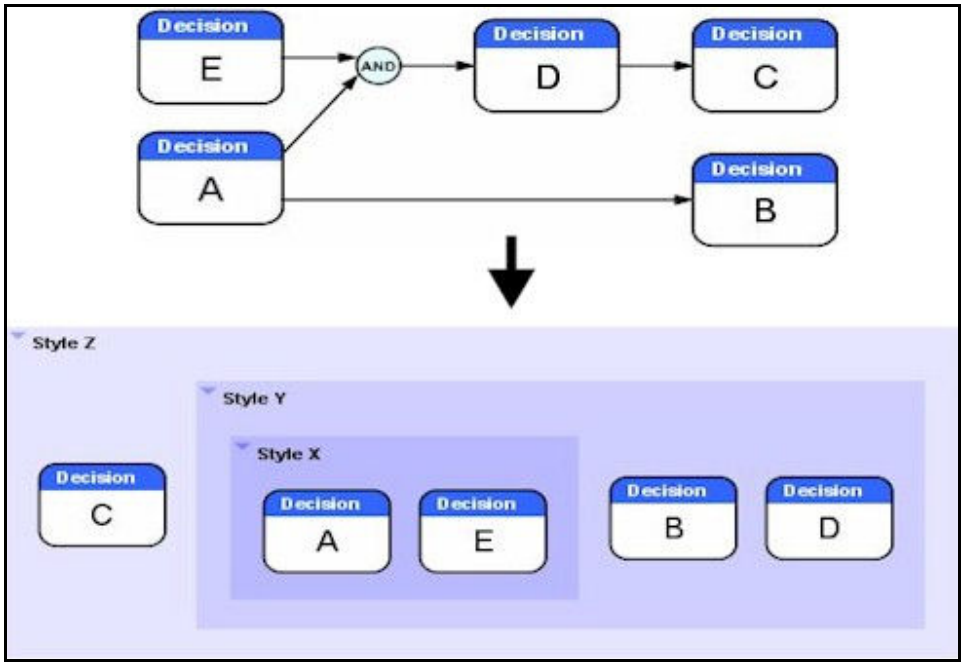

Figure 11: Example of how we draw sequences of architectural decisions

Figure 12 shows a small part of the influence diagram built for REST. The utility nodes were extracted from Chapter 2, and Chapter 4. Each decision represents an architectural style used in the derivation of REST, and each chance corresponds to a reason why a given decision affects a given utility. The chances were extracted from Chapters 2, 3 and 5.

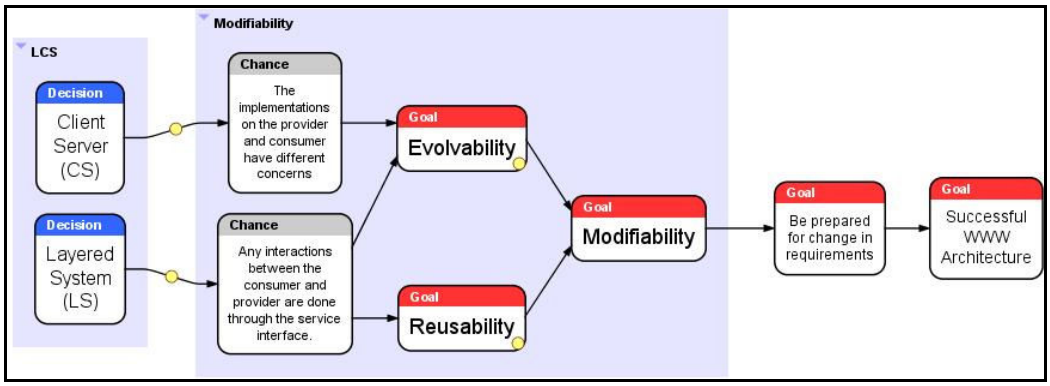

Figure 12: Sample of REST influence diagram

The dashed rectangles in the diagram have the ability to collapse its contents while maintaining the arrows going from the group to the outside. Figure 13 shows the result of collapsing both named rectangles. The little circles that are present in some arrows and boxes denote commentaries. We used them to copy/paste contents from the Dissertation to document nodes and their relationships.

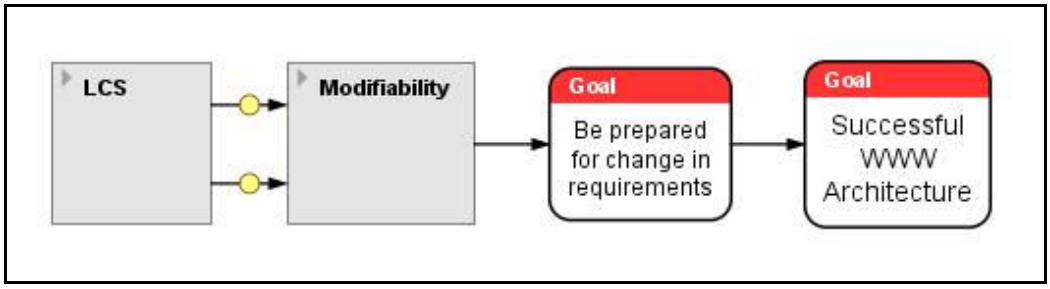

Figure 13: Sample of REST influence diagram, collapsed

Figure 14 shows a condensed influence diagram of REST.

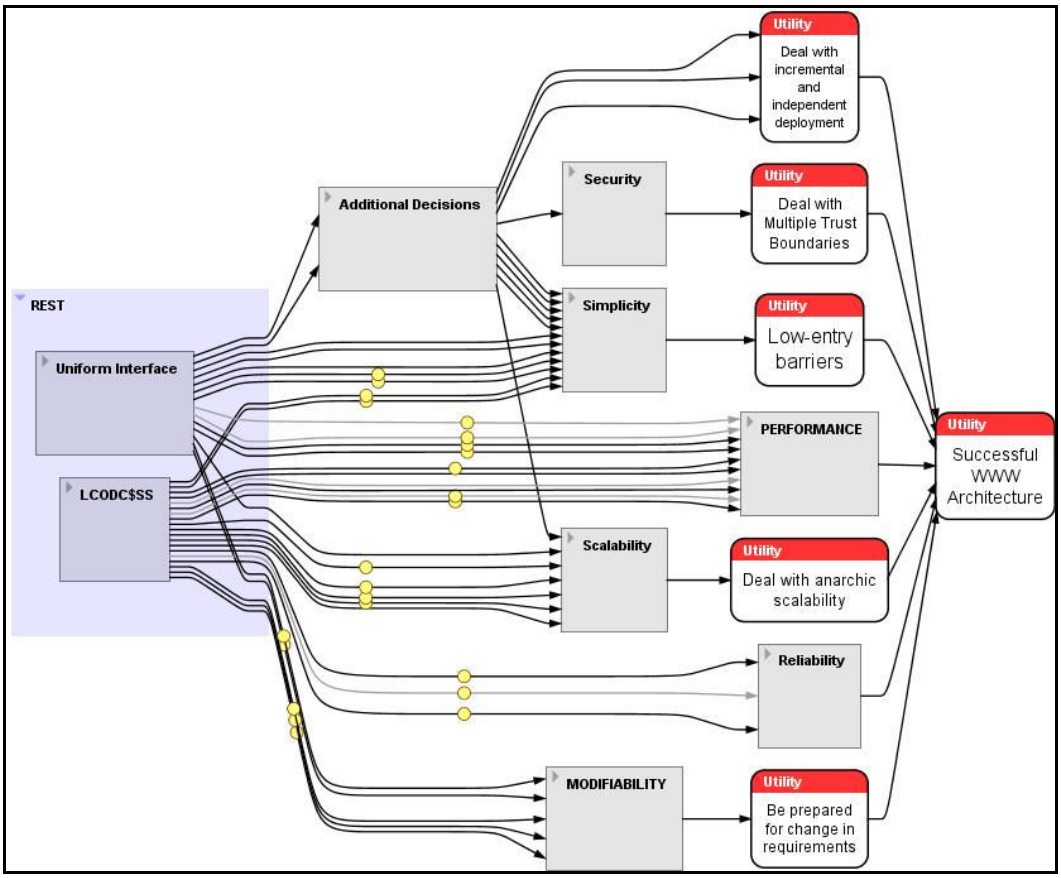

Figure 14: Condensed view of REST influence diagram

In the following paragraphs we comment the process we followed to define the utility, chance and decision nodes of the diagram.

2.7.1 Utilities

If we deleted all the chances and decisions from the diagram and expanded all the utility nodes, we would get a horizontal version of the tree of Figure 15. The figure shows the hierarchy of utility nodes that we inferred from the Dissertation. These utility nodes were extracted from Chapter 2 and 4. Relating general network based software qualities with the goals of the WWW architecture was not

straightforward. Furthermore, while developing the diagram, we decided to eliminate some redundant utility nodes (portability), transform others into chances (visibility), divide some utilities into lower level utility nodes (simplicity), and rearrange the hierarchy of utility nodes (UP performance, Net Performance and Net Efficiency). All these changes were for the sake of making the diagram easier to understand and reuse.

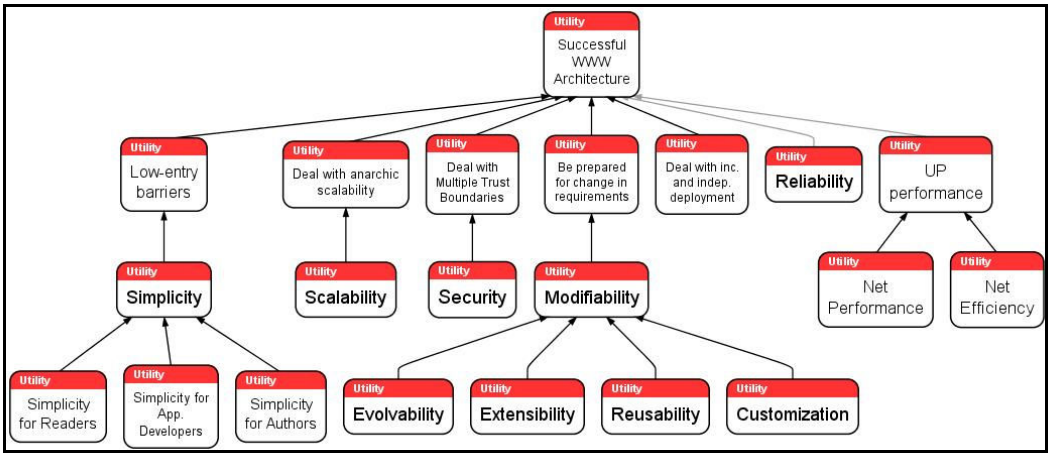

Figure 15: Goals of the standard WWW architecture

2.7.2 Chances

The chances were extracted through an exhaustive revision of Chapters 2, 3 and 5. Once we identified the reason why a decision was related to a utility node, we created a chance between both nodes, or reused an already existent one. In order to maximize reuse of chance, we tried to generalize each chance as much as possible. For example, in Chapter 3 the description of the CSS style includes the following statement: *"Scalability is improved because not having to store state between requests allows the server component to quickly free resources and further simplifies implementation."* From this statement, we extracted the following chance node: "Servers need to maintain little knowledge about clients", which

would later be reused by another decision: “Hypermedia elements are unidirectional”.

2.7.3 Decisions

Initially, every decision node corresponded to one of the styles used in the derivation of REST. However, we soon found out that we had to use finer grained decisions. We then started to divide styles in smaller parts. Figure 16 shows the decisions that compose the Uniform Interface in the REST influence diagram. The decisions that we had to add are: Standard Representations, Standard Operations and Hypermedia as the User Interface.

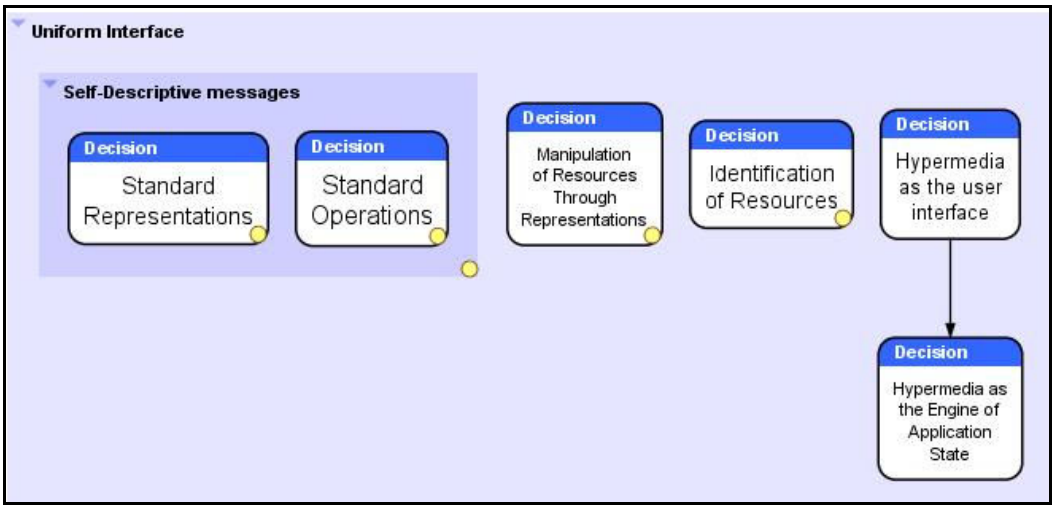

Figure 16: Architectural decisions of the uniform interface

2.7.4 Decisions beyond REST

While developing the diagram, we also extracted some decisions and corresponding chances that were not explicitly explained as parts of REST derivation. Those decisions are shown in Figure 17. Some of them can be considered constraints that can be added to REST to characterize a more restricted architectural style. For example, the decision to define an idempotent operation to request a representation of a resource (i.e. GET in HTTP) could be seen as a constraint that can be added to REST to define a new architectural style. Thus, the influence diagram can facilitate the task of checking what happens if one adds more constraints to REST. All what is needed is to write the constraint as a decision node, find the chances and utilities that it impacts, and reason about its usefulness.

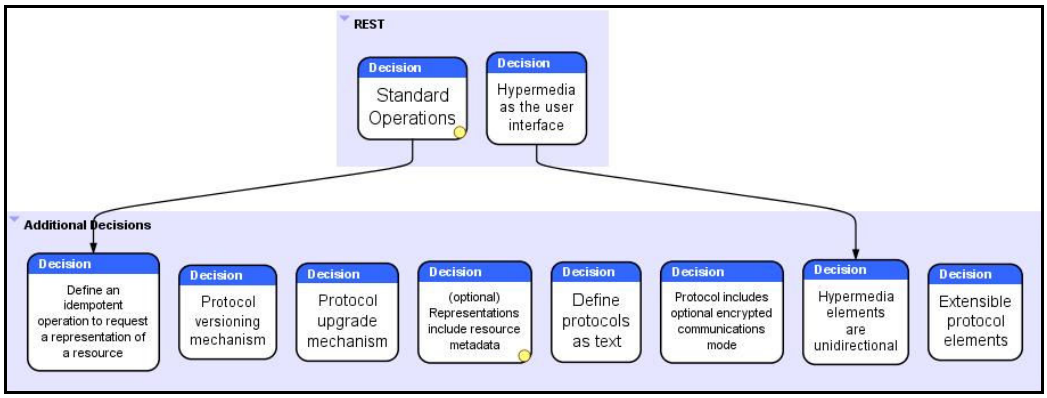

Figure 17: Architectural decisions outside REST

2.8 Lessons learned from the REST Influence Diagram

Some conclusions can be immediately verified by simply looking at the condensed view of Figure 14. Others require further inspection of the diagram:

a) **Simplicity, Scalability, and Modifiability are the main properties of REST.** The goals of Simplicity and Scalability are those that receive more positive effects from REST. They are followed by modifiability and performance, thought performance is also affected negatively by some decisions of REST.

b) **Security is not affected by REST**. One interesting conclusion is that the goal of Security is not tackled directly by REST. It is only treated in a lower abstraction level, and only by one chance: “Protocol includes optional encrypted communications mode” which in turn affects the sub-utility node: “network-level privacy”.

c) **Reliability is partially affected by REST**. In addition, it is interesting to note that although the property of Reliability has four chances extracted from Chapter 2, only two are positively affected by REST, and both comes from the Client-Stateless-Server style.

d) **REST was designed for human users**. Thought REST clients can be robots, it was clear while extracting the chances and decisions that the Author was privileging the case of the users being human. This is manifested, for example, in the chances affecting the goal of Simplicity, that we had to divide into: “Simplicity for Authors”, “Simplicity for Readers”, and “Simplicity for Application Developers”, who are the users introduced in Chapter 4. One practical consequence of this is that, if we would like to make an assessment of how REST induces simplicity in SOA, we should start by classifying the users of SOA and twisting and redistributing the chances related to the sub-utilities of Simplicity in the REST diagram, into the sub-utilities defined for SOA.

2.9 Evaluation

In order to evaluate the applicability of the proposed model, we show how it would be used to respond to the following question, which is relevant among

REST practitioners willing to apply it for web application development: *Considering the recent clarifications* (Fielding, 2008b) *regarding widespread misunderstandings about REST among practitioners of the Resource Oriented Architecture* (Richardson & Ruby, 2007). *What properties are missing in ROA?*

2.9.1 Identifying the missing properties of ROA

The Resource Oriented Architecture is defined in (Richardson & Ruby, 2007) as "a way of turning a problem into a RESTful web service: an arrangement of URIs, HTTP, and XML that works like the rest of the Web, and that programmers will enjoy using". Another term used for the same purpose is Web Oriented Architecture, which is used by some IT consultants to name the application of the WWW standard protocols and proven solutions for the construction of distributed software systems (Hinchcliffe, 2008).

Recently, with the blog posts mentioned before (Fielding, 2009, 2008a, 2008b) it became clear that these approaches are not following all of the REST constraints, so their practitioners started to discuss ways to follow all of the REST constraints and to debate how important is to follow all of them.

The REST influence diagram is a good tool to identify what properties are really missing by not following all the constraints. In order to accomplish this task, we took the REST influence diagram and deleted all Decision nodes that are not relevant to the study. These Decisions are those represented by the LC$SS style (Table 1) and by those architectural decisions used for the development of HTTP and URI but not corresponding to the core of REST (the "Additional Decisions" in Figure 17). The resulting diagram is shown in Figure 18.

.

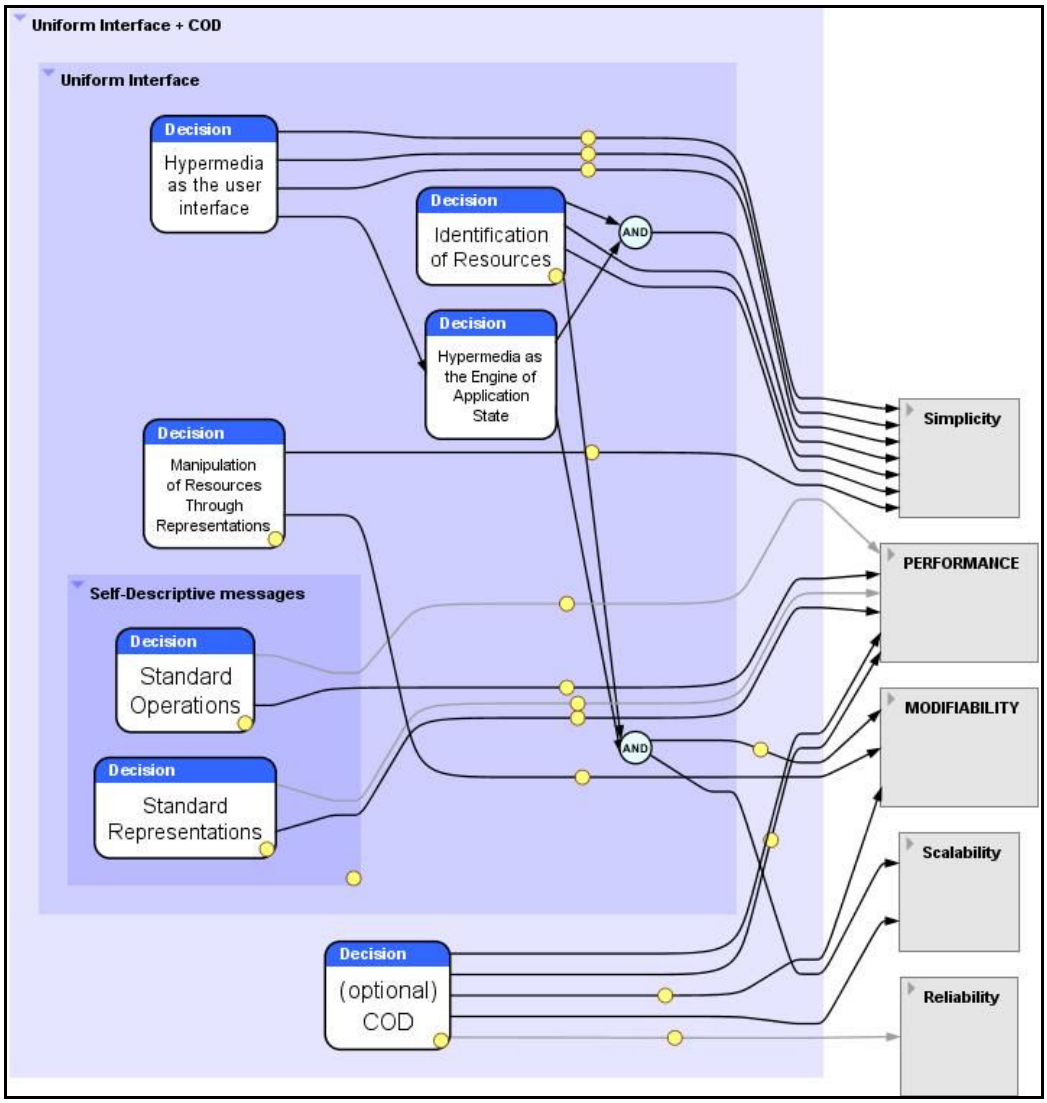

Figure 18: Properties of COD and the uniform interface

ROA lacks only one decision: “Hypermedia as the engine of application state” and places another decision as optional: “Hypermedia as the user interface” (by just recommending that the resource representations should be “connected” by links).

Now, if we focus only in the arrows going from these two decisions we get the diagram of Figure 19. Clearly, the constraint “Hypermedia as the user interface”

was only added to REST as a means to lower the WWW entry barriers for human readers and authors. Thus it is logical that it didn't make much sense for developers thinking about using REST for machine to machine interaction.

On the other hand, "Hypermedia as the engine of application state" affects two chances that would be kept in the diagram even if we instructed the reader to think that the user is now a machine, and these chances impact Scalability and Evolvability. So why did the ROA proponents miss this constraint? Because respecting that constraint in the case of machine to machine integration means that we have to develop representations containing semantic hypermedia, something that would affect negatively the utility of "Simplicity for Application Developers" and "Performance" if the Influence Diagram of REST were made for the goals of machine to machine integration. Thus, it happens that this additional effort is not considered worthwhile when compared to the benefits of Scalability and Modifiability.

Therefore, the answer to the original question is that by not following the hypermedia constraint, the architectures conforming to ROA are less scalable and modifiable than architectures conforming to REST. Exactly how less scalable and modifiable cannot be inferred from the Dissertation. Therefore, the tradeoff of adding these constraints to ROA at the cost of lower simplicity and performance must be the subject of future research.

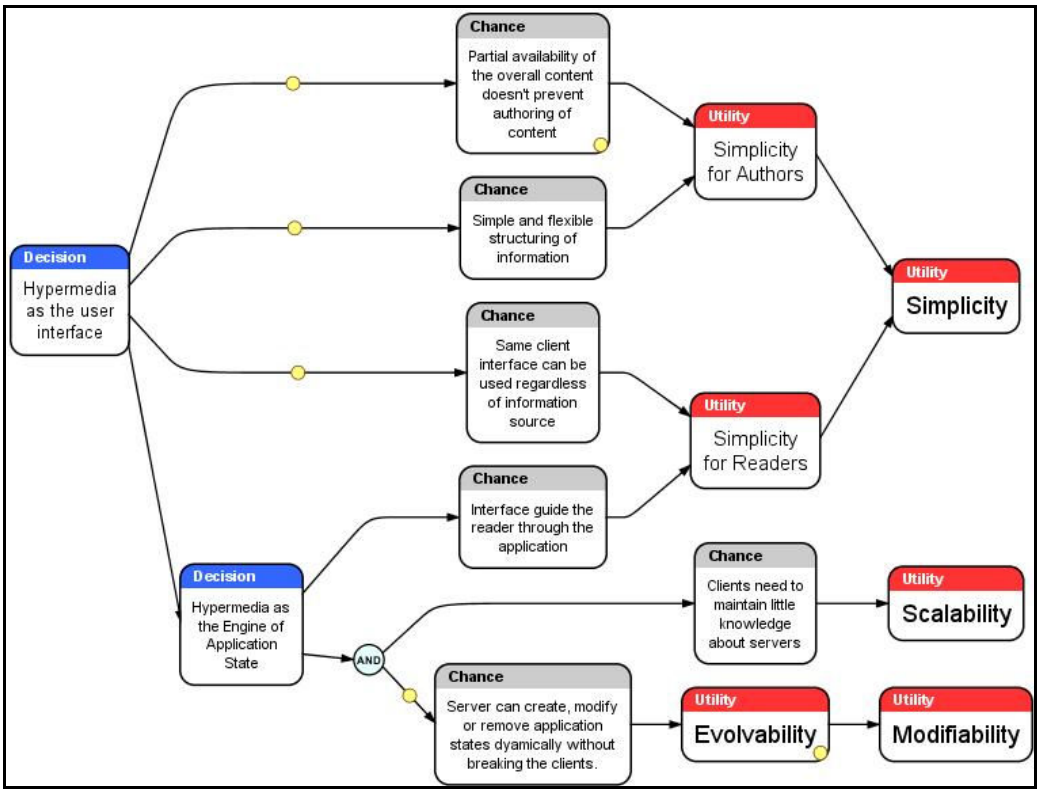

Figure 19: Properties of Hypermedia

2.9.2 Discussion

The answers that could be obtained using the REST influence diagram may not be surprising to REST practitioners, but the use of the influence diagram can guarantee that the response is made considering all the knowledge included in the Dissertation and nothing else. This is one of the most valuable properties of this model. The second valuable property is that the model can be easily extended by adding decisions and chances. For example, one could change both decisions in Figure 19 by one group called “Semantic hypermedia as the engine of application state”, and add to that group decisions like “Machine-readable hypermedia as the client interface” and “Shared semantic data model between Client and Server”,

thus starting to build a new architectural style that reuses part of REST structure and design rationale. In addition, one can reason about applying REST to other problem domains by modifying the hierarchy of utilities to reinterpret the software qualities used in the derivation of REST. For example, if we change “Successful WWW Architecture” for “Successful SOA” in Figure 15, all the other utilities would change their meaning, and we would be forced to change the content of the chances and to re-consider every arrow from decision to chance and chance to utility, but the decisions comprising REST would remain the same.

On the other hand, the example also illustrates some shortcomings of our model. First, the chances are possibly too verbose, and it is hard to define them in such a way that guarantees their reuse with future decisions (like it would be the case of a chance like “lines of code” or “memory size”). Second, if an inexperienced reader reads through the node names, he won’t understand much, unless good comments are provided. One way to solve these problems would be to define primitive architectural decisions, which would be used to model any architectural style. Such effort would improve the understandability of the diagrams, by specifying once the semantics of the decision names, and also the chances that are compatible with each decision.

2.10 Related Work

(Pautasso et al., 2008) use a conceptual framework to compare RESTful APIs with WS-* implementations of web services in three levels of abstraction: principles, conceptual and technology. Using the concept of “architectural decision” and “alternatives” in each of these levels, they conclude that RESTful APIs are good for tactical ad hoc integration because of their simplicity, but that WS-* are better for complex enterprise integration scenarios with longer lifespan and advanced

QoS requirements, because of the alternative architectural decisions one can take using this option. Even thought this is the first paper to compare both approaches extensively, they do it using their own conceptual framework and because of that, they miss important parts of the design rationale of REST. In particular, they didn't give any attention to the constraint of using hypermedia as the engine of application state, which was later pointed out as a key requirement to call an API RESTful (Fielding, 2008b). The REST influence diagram that we proposed in this work would serve as a better framework to compare HTTP, URIs and Hypermedia with SOAP, UDDI, and WSDL.

(Overdick, 2007) presents REST as a manifestation of Resource Orientation, which he describes as an architectural style that is derived from Service Orientation, which is in turn a style derived from the Distributed Objects style. This characterization of REST as a more constrained version of SOA is also used in other publications (Gall, 2008). Although this view of REST is useful to understand intuitively its relationship with SOA, it is too simple to reason about the consequences of making changes to REST, because it hides the architectural decisions and methods that were used to create it. Furthermore, some terms used in this work are also used in the Dissertation with a different meaning. For example, in Chapter 3, Distributed Objects and REST are both architectural styles derived from the Client-Server style. REST is not derived from Distributed Objects.

(Dillon, Wu, & Chang, 2008) describe several architectural styles for implementing service oriented computing, one of those being REST, and they classify the styles by locating them in an "evolutionary cube" with the axis being: "simplification", "loose coupling" and "decentralization". Unfortunately, the styles used are not compatible with those presented in the Chapter 3 of the Dissertation, and the hypermedia constraint of REST is dismissed, instead of considering semantic hypermedia. Therefore, the location of REST in the evolutionary cube

presented in their work may not be accurate. In order to assess the exact place of REST in a cube like that, one should characterize architectural styles for service oriented computing using the same building blocks that were used to characterize REST. These building blocks could be the styles presented in Chapter 3, or the finer-grained architectural decisions proposed in our work.

Another publication that explains SOA as an architectural style is (Lublinsky, B, 2007). Even thought the paper is interesting to understand SOA, the concepts of software architecture are again completely different to those introduced in the Dissertation. For example, there is no notion of components, connectors and data as the elements that are constrained by the constraints that define an architectural style. Therefore it is not possible compare the models of SOA provided in this paper with REST.

Perhaps the ambiguity of the concept of architectural style present across all these works is a consequence of the qualitative approach taken in the Dissertation to characterize and classify architectural styles. This problem was identified in (Mehta & Medvidovic, 2003). In that work the authors propose a set of architectural primitives that are useful to describe all the architectural styles presented in the Dissertation, and they describe the styles using it. As a consequence, they learn that the derivations are not strictly correct. Unfortunately, the model they propose to describe architectural styles is not practical. Following our analysis of section 3, what they did is to introduce a new token model of the set of all the possible configurations of the elements of an architectural style. This model contains more information about the architectural style than, for example, a process view, but it is even more impractical to describe its design rationale. Thus, a more usable approach would be to find “primitive architectural decisions” that could be composed to describe any architectural style, using the generalized type model that we introduced in section 4.

2.11 Conclusions and Future Work

This work has identified and tackled the issues that have caused so many confusions and misleading debates about the REST architectural style. First, we analyzed the Dissertation's background contents from the perspective of modeling theory, as a way to relate the concepts introduced there with common ground terminology. From that analysis we identified the type of models available for REST, described their flaws, and characterized how a practical model of its structure and design rationale would be. Then, we proposed a practical model of REST that combines architectural decisions and extended influence diagram, and populated it with data extracted exclusively from the Dissertation. Finally, we used the diagram to reason about some properties of REST, and showed a sample application of this artifact to respond to a question concerning REST practitioners in the web application development arena.

We believe that this work should be taken into consideration by any researcher interested in studying REST's applicability to different problem domains than that of the WWW architecture.

There are several ways in which this work could be extended. First, the influence diagrams could be modified to provide not only qualitative but also quantitative assessments, thus helping the architect to reason more effectively about the impact of different architectural styles on the desired set of software qualities. Second, a set of primitive architectural decisions could be studied in order to disambiguate the characterization of architectural styles and improve the understanding of the diagrams. Third, modified versions of the REST influence diagram could be done to evaluate the applicability of REST to other problem domains (e.g. SOA).

Finally, a tridimensional version of the proposed practical model, with decisions, chances and utilities distributed in different planes would improve visibility and would allow more space to describe the temporal precedence of the architectural decisions.

3 CONCLUSIONS

This work has identified and tackled the issues that have caused so many confusions and misleading debates about REST architectural style. First, we analyzed the dissertation's background contents from the perspective of modeling theory, as a way to relate the concepts introduced thereby with common ground terminology. From that analysis we identified the type of models available for REST, described their flaws, and characterized how a practical model of its structure and design rationale would be. Then, we proposed a practical model of REST that combines architectural decisions and extended influence diagram, and populated it with data extracted exclusively from the Dissertation. Finally, we used the diagram to reason about some properties of REST, and showed a sample application of this artifact to respond to a question concerning REST practitioners in the web application development arena.

Regarding the list of questions motivated our research; there is only one that wasn't completely addressed:

a) Given that REST is an architectural style designed for the requirements of the standard WWW architecture, why is everybody trying to apply REST to SOA? Shouldn't we instead try to design a custom architectural style for a standard SOA architecture? What are the barriers to accomplish this task?

We believe that researchers and practitioners have been trying to apply REST to SOA instead of designing a custom architectural style for it, because of the following obstacles:

a) There isn't consensus about neither the meaning nor the usefulness of the concepts of software architecture, and architectural styles.

b) Good conceptual models for building architectural styles are missing, and

c) REST was designed after the specification of the WWW standard protocols; therefore, it is not clear that developing an architectural style before the required software architecture is a proven practice.

With the models introduced in this thesis, we hope to be contributing to the progress of the field of software architecture, by overcoming the first two obstacles. The third one will remain as an open issue. Is it possible to design a successful standard architecture for SOA by reasoning about architectural constraints and their impact over a set of desired qualities? Perhaps the only way to design a successful architecture is by trial and error. In any case, concepts and models like the ones presented in this thesis would be required to test that hypothesis.

REFERENCES

Allamaraju, S. (2008). Describing RESTful Applications. Retrieved May 4, 2009, from http://www.infoq.com/articles/subbu-allamaraju-rest.

Altinel, M., Brown, P., Cline, S., Kartha, R., Louie, E., Markl, V., et al. (2007). Damia: a data mashup fabric for intranet applications. En *Proceedings of the 33rd international conference on Very large data bases* (pp. 1370-1373). Vienna, Austria: VLDB Endowment.

API Dashboard: ProgrammableWeb. (2009). Retrieved June 10, 2009, from http://www.programmableweb.com/apis.

Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (1999). *Non-Functional Requirements in Software Engineering*. Springer.

Dillon, T., Wu, C. y Chang, E. (2008). Reference Architectural Styles for Service-Oriented Computing. En *Network and Parallel Computing* (pp. 543-555).

Fielding, R. (2008a, March 22). On software architecture » Untangled. Retrieved June 2, 2009, from http://roy.gbiv.com/untangled/2008/on-software-architecture.

Fielding, R. (2008b, October 20). REST APIs must be hypertext-driven. Retrieved June 2, 2009, from http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.

Fielding, R. (2009, March 20). It is okay to use POST » Untangled. Retrieved June 2, 2009, from http://roy.gbiv.com/untangled/2009/it-is-okay-to-use-post.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. University of California, Irvine.

Floyd, I. R., Jones, M. C., Rathi, D. y Twidale, M. B. (2007). Web Mash-ups and Patchwork Prototyping: User-driven technological innovation with Web 2.0 and Open Source Software. En *Proceedings of the 40th Annual Hawaii International Conference on System Sciences* (pág. 86). IEEE Computer Society.

Flying Logic. (s.d.). Sciral. Retrieved June 1, 2009, from http://flyinglogic.com/.

Gall, N. (2008, November 10). Why WOA vs. SOA Doesn't Matter. Retrieved June 2, 2009, from http://www.itbusinessedge.com/cm/community/features/interviews/blog/why-woa-vs-soa-doesnt-matter/?cs=23092.

Hinchcliffe , D. (2008). What Is WOA? It's The Future of Service-Oriented Architecture (SOA). *Dion Hinchcliffe's Blog - Musings and Ruminations on Building Great Systems*. Retrieved January 11, 2008, from http://hinchcliffe.org/archive/2008/02/27/16617.aspx.

Jansen, A. & Bosch, J. (2005). Software Architecture as a Set of Architectural Design Decisions. En 5th. Pittsburgh, PA, USA.

Jhingran, A. (2006). Enterprise information mashups: integrating information, *simply*. En *Proceedings of the 32nd international conference on Very large data bases* (pp. 3-4). Seoul, Korea: VLDB Endowment.

Johnson, P., Lagerström, R., Närman, P., & Simonsson, M. (2007). Enterprise architecture analysis with extended influence diagrams . *Information Systems Frontiers*, *9*(2-3). doi: 10.1007/s10796-007-9030-y.

Kühne, T. (2006). Matters of (Meta-) Modeling. *Software and Systems Modeling*, *5*(4), 369-385. doi: 10.1007/s10270-006-0017-9.

Lublinsky, B. (2007, September 1). Defining SOA as an architectural style. *IBM Developer Works Technical Library*. Retrieved April 29, 2009, from http://www.ibm.com/developerworks/architecture/library/ar-soastyle/.

Mehta, N. R., & Medvidovic, N. (2003). Composing architectural styles from architectural primitives. *SIGSOFT Softw. Eng. Notes*, *28*(5), 347-350. doi: 10.1145/949952.940118.

Muehlen, Z., Nickerson, J., & Swenson, K. (2004). Developing web services choreography standards—the case of REST vs. SOAP. *Decision Support Systems*, *40*(1). doi: 10.1016/j.dss.2004.04.008.

Overdick, H. (2007). The Resource-Oriented Architecture. En *Services, 2007 IEEE Congress on* (págs. 340-347). doi: 10.1109/SERVICES.2007.66.

Pautasso, C., Zimmermann, O., & Leymann, F. (2008). Restful web services vs. "big"' web services: making the right architectural decision. En *Proceeding of the 17th international conference on World Wide Web* (pp. 805-814). Beijing, China: ACM. doi: 10.1145/1367497.1367606.

Richardson, L. y Ruby, S. (2007). *Restful web services* (p. 446). O'Reilly.

Vinoski, S. (2008a). RPC and REST: Dilemma, Disruption, and Displacement. *Internet Computing, IEEE*, *12*(5), 92-95. doi: 10.1109/MIC.2008.109.

Vinoski, S. (2008b). Convenience Over Correctness. *Internet Computing, IEEE*, *12*(4), 89-92. doi: 10.1109/MIC.2008.75.

Vinoski, S. (2007). REST Eye for the SOA Guy. *IEEE Internet Computing*, *11*(1), 82-84.

Wong, J. & Hong, J. I. (2007). Making mashups with marmite: towards end-user programming for the web. En *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 1435-1444). San Jose, California, USA: ACM. doi: 10.1145/1240624.1240842.

Xu, X., Zhu, L., Liu, Y., & Staples, M. (2008). Resource-oriented business process modeling for ultra-large-scale systems. En *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems* (pp. 65-68). Leipzig, Germany: ACM. doi: 10.1145/1370700.1370718.

COMPLEMENTARY MATERIAL

In addition to this document, a compact disk is provided. The CD includes the REST influence diagram. In order to see this material, it is necessary to install the reader version of the software Flying Logic (http://www.flyinglogic.com/).