Chapter 1
The Essence of REST Architectural Style

Jaime Navon and Federico Fernandez

Abstract There is an increasing interest in understanding and using REST architectural style. Many books and tools have been created but there is still a general lack of understanding its fundamentals as an architecture style. The reason perhaps could be found in the fact that REST was presented in a doctoral dissertation, with relatively high entry barriers for its understanding, or because the description used models that were more oriented towards documentation than to working practitioners.

In this chapter we examine, in a systematic manner, some of the issues about Fielding's doctoral dissertation that have caused so much confusion. We start examining REST as an architecture style as a sequence of architectural decisions. We use then influence diagrams to build a model that allows us to see how the architectural decisions take us from classic architectural styles like client-server and layered-system to REST. The graphical model not only facilitates the understanding of this important new architectural style, but also serves as a framework to assess the impact of relaxing or adding more constraints to it. As a final example we analyze the resource-oriented architecture (ROA) to find out one important constraint that is present in REST is missing in ROA and this has an impact on both scalability and modifiability.

Introduction

REST is usually referred to, as it was originally introduced in the Chap. 5 of the Ph.D. dissertation of Dr. Roy Fielding: an architectural style (Fielding 2000). To fully understand the idea it is necessary to read the full dissertation since REST

J. Navon (✉)
Department of Computer Science, Universidad Catolica de Chile, Santiago, Chile
e-mail: jnavon@ing.puc.cl

E. Wilde and C. Pautasso (eds.), *REST: From Research to Practice*,
DOI 10.1007/978-1-4419-8303-9_1, © Springer Science+Business Media, LLC 2011

rationale cannot be understood without the definitions and concepts of Chaps. 1–3 and the description of the WWW architecture requirements of Chap. 4. The problem is, that even after reading the complete dissertation you might still have questions related more to the real-world implications than to abstract theoretical software engineering issues.

Why is that, in spite of the huge success of REST, there is still so much debate about whether a given service API should be considered REST? What is the difference between REST and Restful? What is the relationship between REST and resource oriented architecture (ROA)?

First there is this slippery thing called Architectural Style. Fielding defines it as 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 conform to that style". This is why, consistent with this definition, he introduces REST precisely trough a set of constraints, namely client–server, stateless, cache, uniform interface, etc.

Explaining REST by introducing constraints associated to a number of primitive architectural styles is just fine for a doctoral dissertation but it leaves a lot of room to interpretation. This is the source of some heated debates about concrete architectures that might be "betraying" the REST principles.

Richardson and Ruby (2007) in their book present what they call a "*ROA*" as a simple set of guidelines that guarantees a RESTful architecture. They make clear though that there are other concrete architectures that may also be RESTful.

Not only is the concept of architectural style is problematic, there is no complete agreement on what is really software architecture. As defined by the Institute of Electrical and Electronics Engineers (IEEE) Recommended Practice for Architecture Description of Software-Intensive Systems (IEEE standard 1471–2000), architecture is "*the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution*". This definition is fairly abstract and applies to systems other than just software. Meanwhile, Fielding emphasizes that software architecture is an abstraction of the run-time behavior of a software system and not just a property of the static software source code.

Bass et al. (2003) define software architecture as "structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them". The plural in structures acknowledges the possibility of more than one structure, each of them conveying architectural information. Some may be more related to the static structure and some more to the dynamic aspects.

We believe that Roy Fielding's dissertation is indeed the key document to dive deeper into the essence of REST; a document that, as Martin Fowler puts forward in his foreword in a recent book (Webber et al. 2010), "is far more often referred to than it is read". We hope that this chapter will contribute to a better understanding of REST and also as a framework to evaluate and discuss new proposals of software architectures and architecture styles.

Architectural Styles and Architectural Properties

As we said before, Fielding definition of architectural style involves architectural restrictions. Furthermore, he suggests that the space of all possible architectural styles can be seen as a derivation tree, where nodes are derived from others by adding new restrictions. Some of these nodes correspond to well known or "basic" architectural styles, whereas others will be hybrid nodes corresponding to combinations or derivations from the basic styles. Traversing the tree from the root to a node would allow us to understand all the design decisions associated to a specific style be it basic or derived. A concrete architecture will adhere more or less to one of these architectural styles depending on how close it is to the cumulative design decisions associated to the corresponding node in the derivation tree. Since each style induces a set of architectural properties, traversing the tree provides us with a good understanding of the architectural properties that our concrete or specific architecture will exhibit once it is implemented.

Fielding uses a qualitative approach to compare some of the most important architectural styles. For each of these styles, identified by a short symbolic name ("Pipe and Filter" is PF, "Client–Server" is CS, etc.), a plus or a minus sign is assigned depending on whether the style under consideration has a positive or negative impact on the software quality, that is on the architecture properties. Sometimes a style may affect a software quality both positively and negatively (because software qualities listed are relatively coarse grained, for the sake of simplicity and visualization).

The impacts of each architectonic style on each quality is presented by Fielding as a table in which there is also information about what styles represent derivations from other styles. 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. Figure 1.1 shows the complete derivation for the REST architectural style. This derivation, the impact table, and some additional explanations are used in the dissertation to describe REST rationale. Table 1.1 is a slightly modified version of the original table. We filled some incomplete cells and added two new rows: one for Uniform Interface (U) and one for REST. The marked cells are those whose signs do not correspond to any possible union of the styles they derive from.

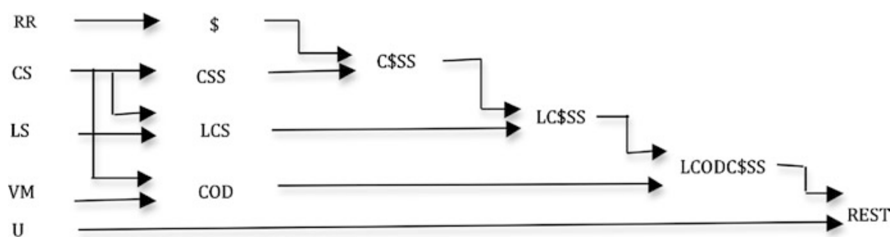

Fig. 1.1 REST derivation tree

Table 1.1 Impact table, extended

Style	Derivation	Net performance	UP perform.	Efficiency	Scalability	Simplicity	Evolyability	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+	++−++	+++	+	+	+	+	+−+	+	+

The REST entry in the table reflects the fact that this architectural style can be derived from several styles (see Fig. 1.1).

Towards a Model for REST

Fielding describes REST by defining the architectural elements (instances of components, connectors and data) present in REST, and using a process view to show some possible configurations of these elements. This model is useful to describe an architectural style, but it is not practical for understanding its design rationale. What is needed is some sort of representation of REST as a set of constraints.

Inspired by the concept of derived styles we explained before, we will describe REST by using the concept of *architectural decision*. 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 gain with the introduction of this concept is that we can now describe an architectural style as a sequence of architectural decisions. This is somehow similar to the derived style approach used by Fielding but making every component of an architectural style explicit. Since the properties of an architectural style become the cumulative properties of its individual architectural decisions, it can be helpful for examining REST design rationale.

Going one step further, what we really want is a model of the REST design rationale that we could manipulate to visualize changes in a concise manner. Here are the ideal requirements for such a model:

- (R1) Visualize and understand how each one of the architectural decisions of REST impacts the set of goals that guided its design.
- (R2) Visualize and understand the changes caused in the induced properties if new architectural decisions are added to REST, or if existing decisions in REST are replaced for others.
- (R3) Visualize the set of alternative architectural decisions for each decision in REST.
- (R4) 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.
- (R5) The model should be a loyal representation of the dissertation idea of REST.

There were several possible options. We could extend the classification framework of Table 1.1 by adding rows at the top grouping 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 much in

terms of understandability. We could instead represent REST as a set of documents, one per architectural decision, containing a description of the decision and the alternatives discarded (Jansen and Bosch 2005) but it wouldn't satisfy requirements R1, R2, and R4.

A better choice is to use a simplified version of *influence diagrams*, a graphical language defined in Johnson et al. (2007). We only need three types of node (*utility*, *chance,* and *decision)* and 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 et al. (1999).

This graphical language allows us to satisfy most of the requirements. The visibility part of R1 and R2 is met using collapsible graphical elements. The understandability part would is met by adding as many kinds of boxes as necessary to trace the impact of a decision over the set of goals. By drawing hierarchies of goals, from the most general ones to those represented by software qualities we can satisfy R4. Finally, extracting only from the dissertation all the knowledge used to define the elements of the diagram and their relationships, and documenting all these definitions with comments in the diagram we satisfy R5. 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.

Analysis of REST Trough Influence Diagrams

Although not necessary, it is nice to use a software tool to build the diagrams. We used *Flying Logic*[1] flexible visual modeling tool. This is a commercial product but there is a free reader available so people who only want to read de diagrams do not need to buy the full product.

The influence diagram nodes corresponding to utility, decisions and chances were represented by tagged boxes (Fig. 1.2). The little circles present in some arrows and boxes denote commentaries (we used them to copy fragments from the dissertation). A black arrow, between a decision and a chance, means that the decision has a positive causal relationship with the chance whereas a grey arrow means that the decision has a negative causal relationship with the chance.

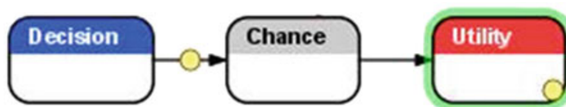

Fig. 1.2 Notation of the influence diagrams

[1] http://www.flyinglogic.com

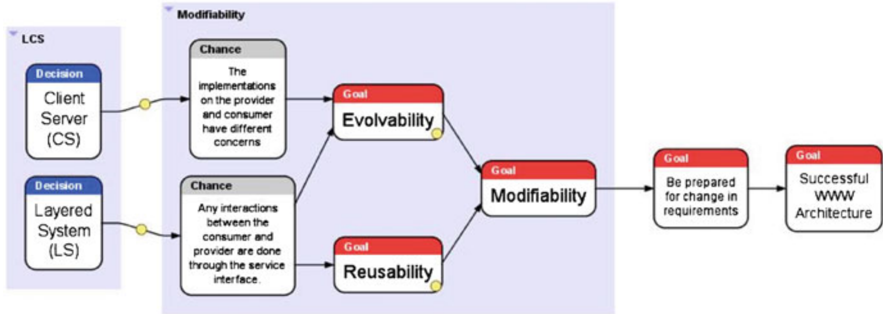

Fig. 1.3 Sample of REST influence diagram

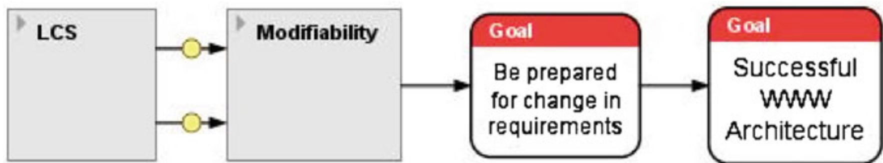

Fig. 1.4 Sample of REST influence diagram, collapsed

An arrow between a chance and a utility can only be black and denotes a positive causal relationship between the chance and the utility node. An arrow between two utilities means that the first utility is a child of the second one. Arrows between decisions were used to denote temporal precedence.

Because of limitations in the tool, we do not use arrows to connect decisions. Instead, we describe the sequence of decisions that defines REST as a hierarchy of decision sets. This improves visibility at the cost of reduced understandability. 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 could not know which was done before the other, so he would have to get this information from another source (e.g. comments).

Figure 1.3 shows part of the REST influence diagram. Each decision represents an architectural style used in the derivation of REST, and each chance explains why a given decision affects a given utility. The dashed rectangles in the diagram can collapse its contents while maintaining the arrows going from the group to the outside (Fig. 1.4 shows the result of collapsing both named rectangles).

A more complete influence diagram for REST in the condensed mode is presented in Fig. 1.5. If we delete all the chances and decisions from the diagram and expand all the utility nodes, we get a horizontal version of the tree (Fig. 1.6).

To produce the chances we performed an exhaustive revision of the dissertation. Once we identified why a decision was related to a utility node, either we created a new chance between both nodes or we reused an already existent one. In order to maximize the reuse of chances, we tried to generalize each chance as much as possible.

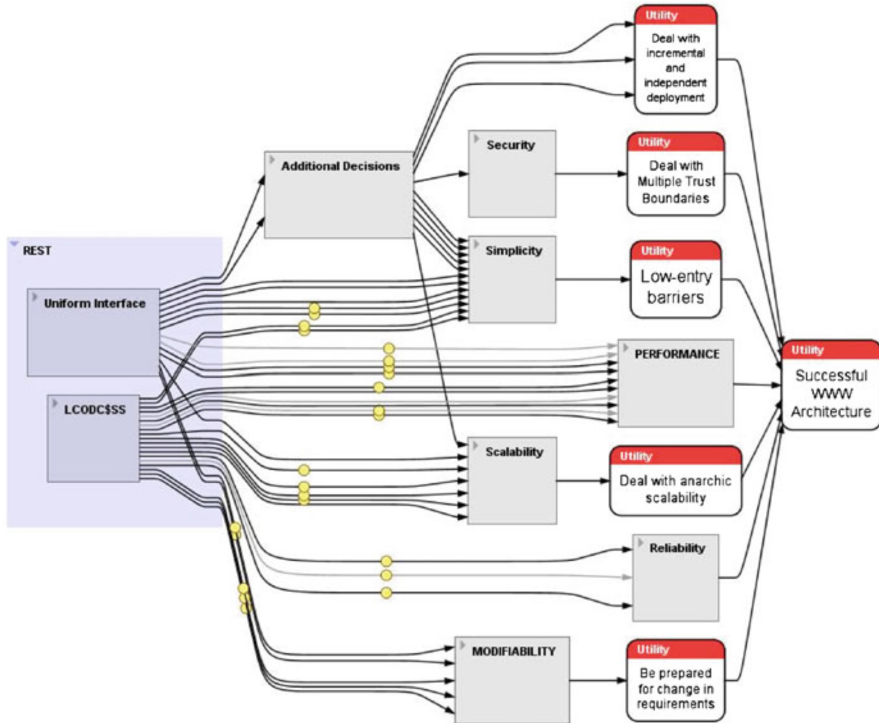

Fig. 1.5 Condensed view of REST influence diagram

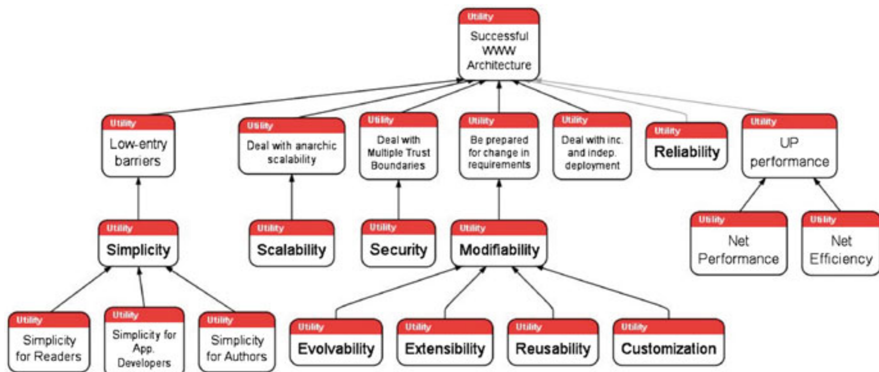

Fig. 1.6 Goals of the standard WWW architecture

Relating general network based software qualities with the goals of the WWW architecture is not straightforward. Furthermore, to make the diagram easier to read and understand, we eliminated some redundant utility nodes (portability), transformed others into chances (visibility), divided some utilities into lower

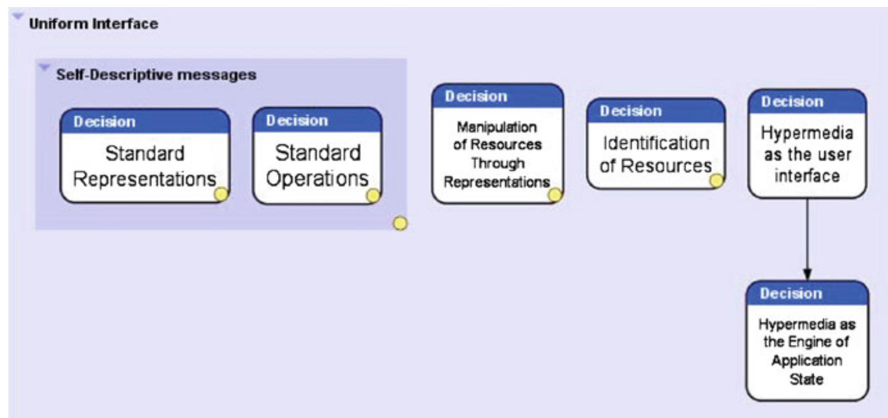

Fig. 1.7 Architectural decisions of the Uniform Interface

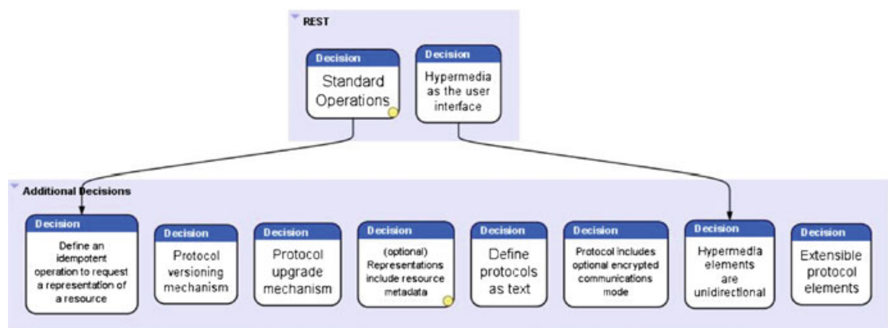

Fig. 1.8 Architectural decisions outside REST

level utility nodes (simplicity) and rearranged the hierarchy of utility nodes (UP Performance, Net Performance and Net Efficiency).

Initially, every decision node corresponded to one of the styles used in the derivation of REST, but then we decided to include decisions of lower granularity by dividing some styles into smaller parts.

Figure 1.7 shows the decisions that compose the Uniform Interface in the REST influence diagram. The decisions we included are Standard Representations, Standard Operations, and Hypermedia as the User Interface.

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 Fig. 1.8. 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 X to request a representation of a resource could be seen as a constraint to be added to REST defining a new architectural style. Any software architecture conforming to X would also be an instance of REST.

The influence diagram then, can facilitate the task of checking what happens if we add constraints to REST. All that is needed is to write the constraint as a decision node, find the chances and utilities that it impacts, and reason about its usefulness.

The coding of the REST architectural style into the diagrams we presented before allows us to reason in a much more easy and precise way. Here are just a few observations:

- Simplicity and scalability are the two goals that receive more positive effects from REST. Modifiability and performance follow although the last one is also affected negatively by some decisions.
- Security is not addressed 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”.
- Although the property of Reliability has four chances, REST affects positively to only two of them, both coming from the Client–Stateless–Server style.
- It is clear that Fielding was thinking in human users rather than bots. This is manifested, for example, in the chances affecting the goal of Simplicity that had to be divided into: “Simplicity for Authors,” “Simplicity for Readers,” and “Simplicity for Application Developers”. One practical consequence of this is that if we want to make an assessment of how REST induces simplicity in SOA, we should start by classifying the users of SOA and redistributing the chances related to the sub-utilities of Simplicity in the REST diagram, into the sub-utilities defined for SOA.

ROA Under the Magnifying Glass

The ROA as defined in Richardson and Ruby (2007) is “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, it became clear that some of these approaches were not following all of the REST constraints, so practitioners started to debate, not only about ways to follow those constraints, but also about how important was to follow all of them.

To further test the applicability of our REST model, we will try now to provide an answer to a recent question that has been subject of a strong: *Considering the*

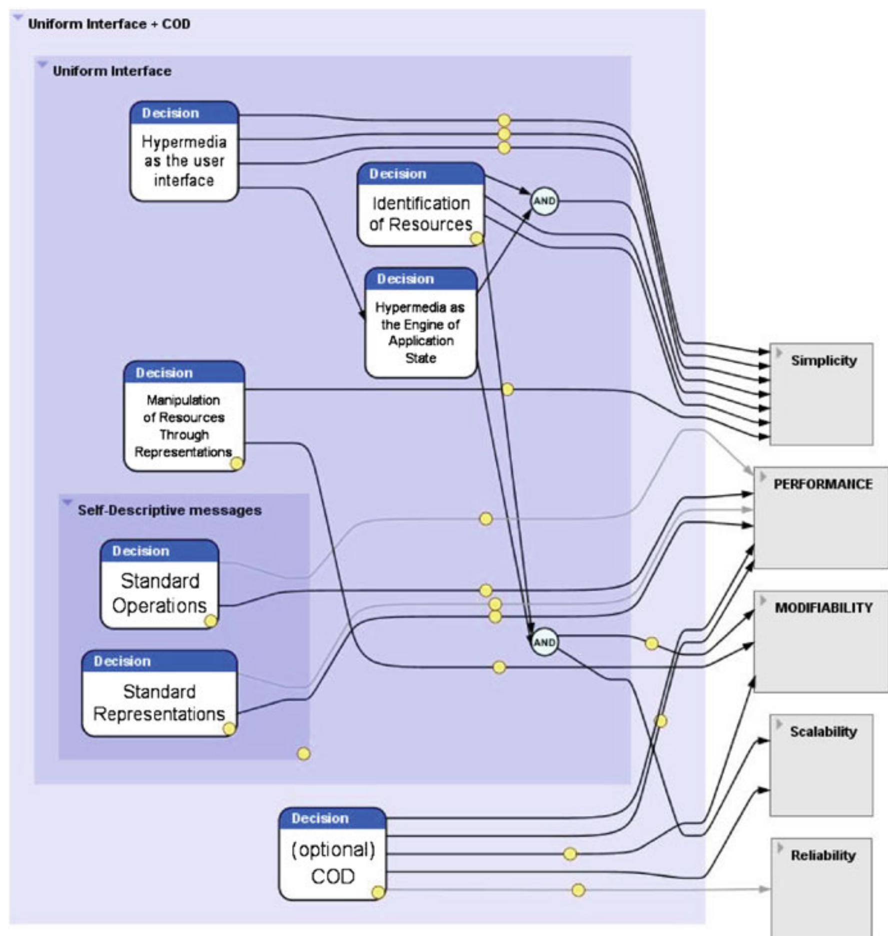

Fig. 1.9 Properties of COD and the uniform interface

recent clarifications[2,3,4] *regarding misunderstandings among practitioners of ROA, what properties are missing in ROA?*

The Influence Diagram of REST is a useful tool to identify what properties are lost by not following all the constraints. To accomplish this task, we take the REST influence diagram and delete all decision nodes that are not relevant. These decisions are those represented by the LC$SS architectural style and by those decisions used for the development of HTTP and URI that do not correspond to the core of REST. The resulting diagram is shown in Fig. 1.9.

---

[2] http://roy.gbiv.com/untangled/2008/on-software-architecture

[3] http://roy.gbiv.com/untangled/2009/it-is-okay-to-use-post

[4] http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

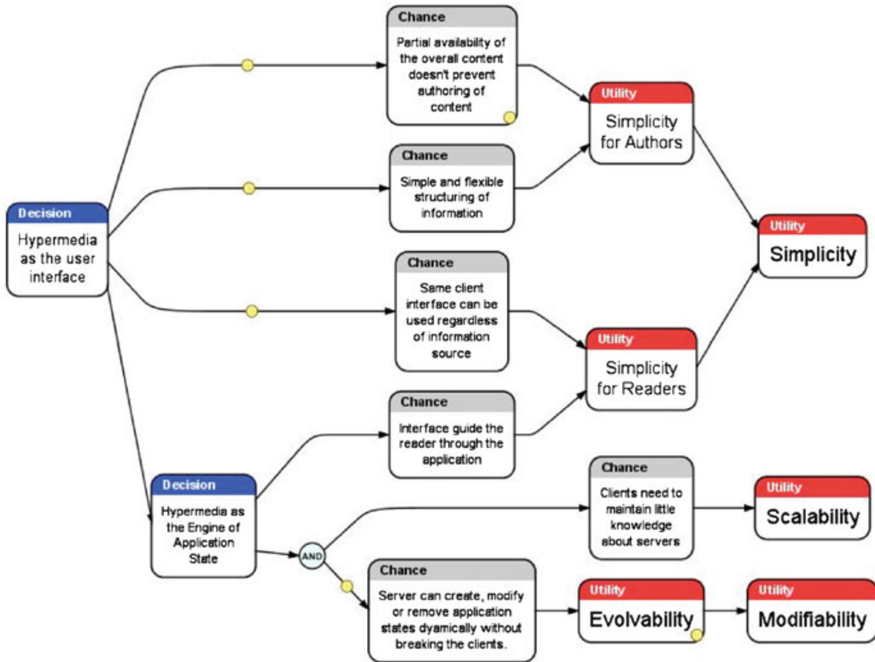

Fig. 1.10 Properties of Hypermedia

ROA lacks only one decision: "Hypermedia as the engine of application state". There is also an optional decision: "Hypermedia as the user interface" (resource representations should be connected by links).

If we focus only in the arrows going from these two decisions we get the diagram of Fig. 1.10. Clearly, the constraint "Hypermedia as the user interface" was added to REST as a means to lower the WWW entry barriers for human readers and authors. This explains why it didn't make much sense for developers thinking rather in 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 think that the user is a machine, and these chances impact Scalability and Evolvability.

So, why did the ROA proponents are ready to relax this constraint? Because in the context of machine-to-machine integration, it may require representations containing semantic hypermedia and this in turn would negatively affect the utilities "Simplicity for Application Developers" and "Performance". The additional effort is not considered worthwhile when compared to the benefits of Scalability and Modifiability.

The answer to the original question, thus, is that by not following the hypermedia constraint, the architectures conforming to ROA are indeed less scalable and modifiable than architectures conforming to REST.

We do not know exactly how less scalable and modifiable would be. The tradeoff of adding these constraints to ROA at the cost of lower simplicity and performance must be the subject of further research. In fact, one interesting research line could explore a way to modify the influence diagram technique to provide not only qualitative but also quantitative assessments, to help the architect to reason more effectively about the impact of different architectural styles on the desired set of software qualities.

The result obtained through REST influence diagram may not be surprising to REST practitioners, but now we can guarantee that it is founded in the knowledge included in the dissertation and nothing else. This is one of the most valuable properties of this model. A second valuable property is that adding decisions and chances can easily extend the model. For example, one could change both decisions in Fig. 1.10 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.

References

Bass, L., Clements, P., and Kazman, R. (2003) Software Architecture in Practice, 2nd Ed., Addison Wesley Professional, Reading, MA, USA.

Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J. (1999) Non-Functional Requirements in Software Engineering. International Series in Software Engineering, Vol. 5, Springer, Berlin, Heidelberg, New York.

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

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 11th, 2008, http://hinchcliffe.org/archive/2008/02/27/16617.aspx.

Jansen, A. and Bosch, J. (2005) Software Architecture as a Set of Architectural Design Decisions. Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture, WICSA '05, Pittsburgh, PA, USA.

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

Richardson, L. and Ruby, S. (2007) Restful web services. O'Reilly Media Inc. USA.

Webber, J., Parastaditis, S., and Robinson, I. (2010) Rest in Practice, O"Reilly Media Inc., USA.