Department of Computing & Mathematics

REST for Real-Time, Cloud Based, Collaboration

By

Michelle Concannon

01st September 2014

Research Supervisors:
Dr John Healy
Dr Sean Duignan

This project is submitted in partial fulfillment of the requirements for the award of Master of Science (Computing)

Abstract

'Although collaborative technology solutions have been around for many years they are kind of like Olympic competitors with a slew of silver medals but no golds: high achievers that have yet to achieve their highest aspirations'

- Mary Hamilton, Alex Kass, Allan E. Alter, Accenture Technology Labs [2013]

The case study at the center of this research proposes a cloud based software architecture & delivery model, partial design & implementation that can be extrapolated & used as a tool for new & existing players in the collaboration industry to satisfy a set of commonly desired system requirements.

The race to take gold in the enterprise collaboration market is on & all evidence suggests that the winner will deliver a set of integrated collaboration workloads that can be offered as a cloud hosted solution leveraging the key technology enablers explored in this research.

Dedication

To the teachers of life I have been so fortunate to learn from.

My sincere thanks.

*

There is no end to education.

It is not that you read a book, pass an examination and finish with education.

The whole of life, from the moment you are born to the moment you die, is a process of learning - Jiddu Krishnamurti

*

You don't learn to walk by following the rules.

You learn by doing & falling over - Richard Branson

Table of Contents

Chapter One

1. INTRODUCTION & THESIS STRUCTURE

According to the International Data Corporation, "the ICT industry is in the midst of a once every 20–25 year shift to a new technology platform for growth and innovation" (Gens, 2013). They refer to it as 'the 3rd platform', which is built on mobile devices & apps, cloud services, mobile broadband networks, big data analytics and social technologies. By 2020, when the ICT industry reaches $5 billion, the IDC projects that at least 80% of the industry's growth will be driven by these 3rd platform technologies, an explosion of new solutions built on the new platform, along with rapidly expanding consumption of all of the above in emerging markets.

This thesis describes a research project in the area of REST & it's role as one of the key technology enablers in the proliferation of mobile & cloud computing practices across the enterprise & consumer-focused collaboration industry.

The research hypothesis proposes that for any cloud delivered, real time, mobile collaboration solution to proliferate to a ubiquitous level, it should be designed around an open, intuitive & REST based API platform at it's inception. Crucial however, is reaching a pragmatic balance between strict adherence to RESTful API constraints & delivering the performance requirements inherent in any real time collaboration platform.

1.1 The Mobile Wave

'The future of work is one where employees become as demanding and innovative as they are as customers. If the technology they have at work is slower, heavier and so locked down that they are unable to access the tools and apps they find useful, the likelihood is that employees will start to use their own better technologies to get the job done' (Millard & Gillies, 2013, p.1).

Mobile computing is a truly disruptive technology trend that has resulted in major changes to business, society & economics. It has yielded non-linear effects on so many levels & at such a grand scale that it is difficult to quantify (Saylor, 2012). Recent findings by a leading industry analyst reveal that mobile device shipments

break 2.4B units in 2014 as show in Figure 1.1 below, while tablets are projected to overtake PC sales in 2015 (Gartner, 2014).

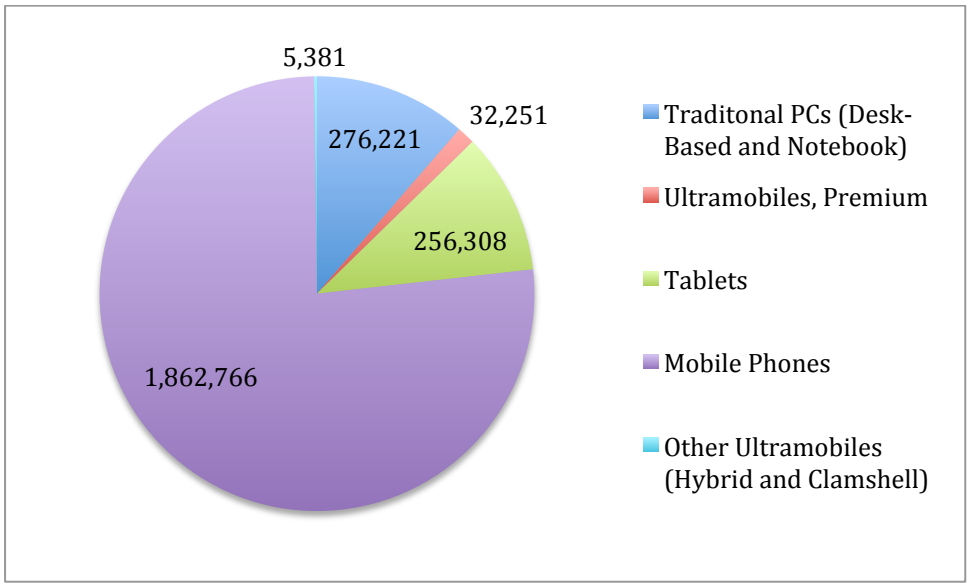


Figure 1.1: 2014 Device Shipments

The real significance of the shift to mobile computing is not about bringing phone calling to handheld computers, rather the establishment of a dramatic new model for applications (the app), a whole new eco system in which those apps get created & distributed (app stores), & a whole new user interface experience (multi-touch). They are the least expensive 'computers' with the lowest cost applications, making them affordable, for the first time in many cases to the majority of consumers. And critically, because mobile devices are also phones, the pervasiveness of compute has exploded, as devices literally exist in every pocket, purse or backpack.

Consumers have become accustomed to the convenience of the delivery model these new software applications provide as new features are seamlessly rolled out to their personal mobile devices via non invasive update notifications being pushed on a daily basis.

The impact of this consumer driven model has also had profound effects in the enterprise domain as users expectations rise with respect to the range of devices available for them to collaborate on, the experiences that are delivered to those devices as well as the pace & ease with which new features get rolled out. Employees

are bringing their own devices to work & expecting full access to both business & personal applications at all times.

IT departments & CTO offices across the industry strive to respond to this trend by replicating the consumer based delivery model at a similar velocity, while maintaining a level of control over such deployments appropriate to enterprise in order to maintain security compliance. The investment around support for the Bring Your Own Device (BYOD) model within traditionally PC oriented enterprises is one concrete piece of evidence of these efforts as detailed in a recent whitepaper published by BT & Cisco (2013).

Productivity gains associated with 'always on' mobile computing & the atypical nature of a 9-5 day for enterprise employees (Millard & Gillies, 2013), are all key drivers toward the shift in investment to mobile first development & the delivery of cloud hosted applications to support the evolving trend.

1.2 Mobile Cloud Computing

Cloud computing has been the topic of research for quite some time now. It is a model for enabling convenient, ubiquitous, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, applications, and services including security, application development, etc.) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Tuli, Hasteer, Sharma & Bansal, 2013).

Cloud delivered applications typically strive to achieve 5 unique properties: multi-tenancy, elasticity, infinite horizontal scale, metrics driven & continuous delivery, all of which brings significant business benefits (Rosenberg, 2014).

On the other hand, the increasing technological enhancement in the field of mobile computing introduced above, is demanding pervasive and ubiquitous computing (PUC), which has given birth to the idea of integrating the concept of Cloud with mobile computing. The term "mobile cloud computing" (MCC) was introduced not long after the concept of "cloud computing" launched in mid-2007 (Hoang, Lee, Niyato, & Wang, 2013).

Briefly, MCC provides mobile users with data processing and storage services in clouds. The mobile devices do not need a powerful configuration (e.g., CPU speed and memory capacity) since all the complicated computing modules can be processed in the cloud. This is powerful in the sense that it overcomes traditional mobile constraints that include processing power & battery life, substituting this instead with the illusion of infinite computing resources. Mobile applications leverage this IT architecture to generate the following advantages:

- Extended battery life
- Improvement in data storage capacity and processing power
- Improved synchronization of data due to "store in one place, access from anywhere" policy
- Improved reliability and scalability
- Ease of integration

With the architectural foundations laid for the growth & innovation projected with the 3rd platform, the software design patterns that enable a developer realize the benefits of a combined mobile & cloud centric architecture, become paramount to unlocking the ultimate goal of unparalleled business growth.

1.3 Representational State Transfer (REST)

Offloading of mobile data processing & storage services to the cloud opens the question of what distributed communication protocol or standards based Application Programming Interface (API) to invest in in order to:

1. Induce the desired cloud / mobile compute properties described above
2. Optimize for application, device & network performance
3. Lower the barrier to development via intuitiveness of API use

Designing an intuitive, standards based API that is easily understood & accessible by the developer community, is one of the most important things a business can do to increase the value of the services it provides. By having an API, the service & core application has the potential to become a platform from which other services can grow as software developers, internal & external to the organization, design & integrate products that are powered by its service. Huge tech companies that have

applied this rule have achieved tremendous success: Facebook, Twitter, Google, GitHub, Amazon, Netflix. None of them would be nearly as big as they are today if they hadn't opened up their data via API to support viral developer integrations.

There have been numerous service oriented architecture API frameworks released over the past two decades, explored in chapter 3, but the cloud-computing world has converged on REpresentational State Transfer (Fielding, 2000) as the architectural style of choice.

By defining architectural constraints such as stateful resources, stateless interactions, global identification of unique resources, uniform interfaces & multiple resource representations, the REST style promotes the design of resource-oriented architectures. These architectures are characterized by their intrinsic interoperability, loose coupling, high scalability & flexibility (Gambi & Pautasso, 2013), all of which are key to delivering the desired characteristics of mobile & cloud computing.

The REST architectural framework outlined in detail in section 2.4 is just that however, a framework. The scope for interpretation is vast & misuse of the guidelines to create a poorly designed 'RESTful' API has the potential to create significant confusion, churn & impact the productivity of consuming clients as well as negatively impact network performance. However, applied effectively & pragmatically, REST based API's can unlock the desired business value inherent in the emergence of the 3rd platform.

1.4 Research Contribution

This research attempts to identify the right balance of pragmatism & purism when designing a RESTful API for real-time, collaboration applications.

It does so by researching & investigating the representational state transfer (REST) software architectural style & then what it means on a practical level to adhere to its constraints.

It considers the real world implications of strict adherence to applying REST

constraints to a collaboration centric API by way of comparative analysis & in doing so, it demonstrates & makes recommendations about the trade offs to consider between often conflicting, success metrics:

- Strict adherence to RESTful API principles to achieve purity of design
- Ease & intuitiveness of use for consuming clients & 3rd party developers
- Optimization of network & application performance

The principle output of this research is a set of data points, API documentation & sample Java code that forms the basis of a comparative analysis of two different approaches to delivering a subset of requirements defined for a fictitious collaboration company, ACME Inc.

It may serve as a framework for new collaboration centric products or startup's seeking to expose a best in class API that is optimized for cloud deployment, accelerated developer integration, ease of use & network performance.

The sample code, developed in Java, can be referenced as a guide to achieve a working level understanding of REST & the implications of each model from both a client & server standpoint as such, the rationale behind the conclusions outlined in Chapter 6.

1.5 Thesis Structure

The remainder of this document is structured as follows:

Chapter Two – reviews the current research literature available in the general area of Distributing Computing, Cloud Computing & REST with a view to helping the reader understand the relevance of REST API design in the context of distributed & cloud computing.

Chapter Three – describes the research framework used. This includes a description of the research methodology & how the proposed application of RESTful principles was tested & evaluated.

Chapter Four – takes a look at the collaboration industry in 2014 & introduces ACME Inc, a fictitious collaboration organisation with a set of common collaboration requirements on which the case study is built. It then deep dives into the exploration of a cloud based architecture & cloud deployment model appropriate for Acme as well as details the specifics of a Java-based RESTful API implementation that delivers a subset of Acme's collaboration requirements. This includes detail of the overall application architecture & links to source code incl. JavaDoc.

Chapter Five – provides a substantive evaluation of both the RESTfulness of the API implemented & the performance it achieves & offers a counter proposal that is less RESTful in design but demonstrated to be more performant.

Chapter Six – outlines the conclusions of this research project & identifies avenues of possible future research.

In addition a link to a public git repo is included that contains the source code of all components of the prototype application, including a README. Javadoc is also included to describe the API delivered in more detail & finally, two JMeter test plans that were the basis on which the performance evaluation was completed & the detailed reports generated by that exercise.

Chapter Two

2. LITERATURE REVIEW

Due to the fact that this project is partially research based, a number of texts & recent literature were consulted across the relevant subject areas to attain a good understanding of the evolution of computing models over the past three decades & how that progression has led to the establishment of the 3rd platform.

Section 2.1 reviews the literature surrounding the fundamental principles of distributed computing & the pertinent criteria that any distributed software solution must adhere to. These core-computing principles remain unaffected by an ever-evolving technology industry. This section also provides an overview of the evolutionary changes that have occurred in parallel, distributed, and cloud computing over the past 30 years to set further context for the relevance of the research.

Section 2.2 presents the findings from a detailed literature review of what it means to operate within a 'cloud computing' model, what the benefits of cloud computing are & why RESTful API's are a key enabler for cloud computing.

Section 2.3 comprises of a review of the key REST constraints Fielding (2000) describes. These constraints form the basis on which the research hypothesis was formed & are the foundation for the case study that follows in chapter 4.

2.1 Distributed Computing

"Someday soon you will look into a computer screen & see reality. Some part of your world – the town you live in, the company you work for, your school system, the city hospital – will hang there in sharp color image, abstract but recognizable, moving subtly in a thousand places. This Mirror World you are looking at is fed by a steady rush of new data pouring in through cables. It is infiltrated by your own software creatures, doing your own business."

- DAVID GELERNTER, DEPT OF COMPUTER SCIENCE, YALE UNIVERSITY

The quotation above is the preamble of a series of papers published in the mid nineties (Clarke, Stikeleather & Fingar, 1996). Less than 10 years after that publication a series of consumer oriented social network & collaboration applications were launched to realize the abstraction by the worlds leading software companies. Today, the vast majority of millennials have difficultly recalling a world without Facebook, Twitter, Google Maps & real time collaboration applications such as FaceTime, Skype, Google Hangouts & WhatsApp. All of these real world instantiations of Gelernter's projection are built atop of highly scalable & performant distributed systems.

The advances in distributed computing & interconnected networks have made possible efficiently & cost-effectively linking people, isolated systems, computing resources & information across the consumer industry & across business units in an organization. It has also contributed significantly to the domain of enterprise integration to bring together large islands of isolated heterogeneous & complex systems (Qiu, 2013).

This section attempts to answer the following questions from the literature reviewed:

- What is distributed computing & how has it evolved?
- What are the key issues involved in designing distributed systems?
- What is REST & how can 'properly' designed RESTful API's address these issues? Cloud computing is emphasized as the latest instantiation of distributed computing architecture within the industry.

2.1.1 Definition, Evolution & Programming Models

A source of confusion in the field of distributed computing is the lack of a universal vocabulary. For the purposes of this research we will use the following definitions taken from Liu (2004).

- A uni-processor, or monolithic computing makes use of a single central processing unit (CPU) to execute one or more programs for each application. The computer is not connected to any network, & thus must only use those resources within it's immediate access.

- A distributed system is a collection of independent computers, interconnected via a network, that are capable of collaborating on a task. Computers are considered independent if they do not share memory or program execution space.
- Distributed computing is computing performed in a distributed system. Computer programs running on independent computers, collaborate with each other to deliver network based services & applications.

Different from parallel computing, each process in distributed computing has its own private memory (or distributed memory), where data for computations is exchanged by passing data between the distributed processes. Figure 1 below illustrates the basic concept.

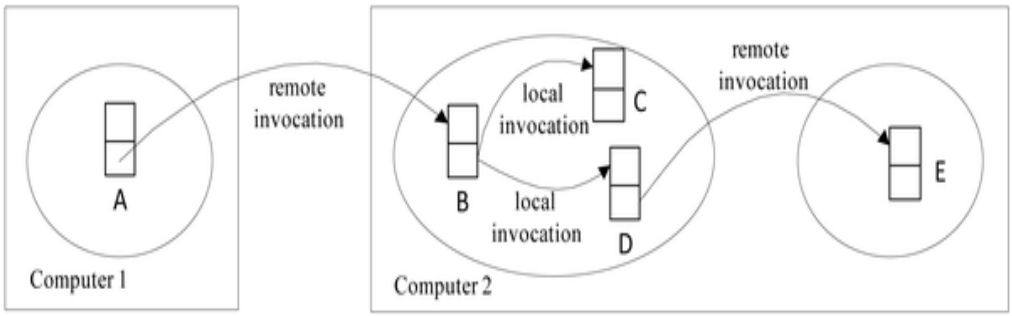


Figure 2.1 A schematic view of local & remote services [Qiu, 2013]

The advantages of distributed computing have been widely reviewed & documented (Birman, 2005, Mahajan & Shah 2013) & several key technology transitions have facilitated the establishment of distributed computing as the norm.

The appearance of the World Wide Web in the 1980's was the defining moment in the transition from monolithic computing to distributed computing. The history of the evolution & growth of the Internet as the first instantiation of a complex distributed network is well documented by Hafner & Lyon (1996).

The literature reviewed for this section describes the transition to distributed computing & it's characteristics from several different perspectives.

Liu (2004) enumerates the following as the key drivers for the migration.

- The affordability of computers & availability of network access

- Resource sharing
- Scalability
- Fault tolerance

Birman (2005) describes the characteristics of a distributed system within the context of the basic building blocks of processes & messages. He considers the following six components of a distributed computing system:

- Communication technology i.e. hardware support for message passing
- Basic transport & network services e.g. the IP protocol
- Reliable transport software & communications support e.g. TCP
- Middleware incl. software tools, utilities & programming languages
- Distributed computing environments i.e. general-purpose tools from which distributed systems can be constructed e.g. Java Enterprise Edition (J2EE) & Microsoft's .Net framework
- End User Applications

The first three of these components pertain to communication channels that rely on connectivity methods. The latter three start to align more closely with the programming models, tools & environments used for developing distributed applications.

Qiu's (2013), business oriented publication takes a different perspective but builds on those two themes. He looks at the transition within the context of associated connectivity methods & programming models. He summarizes how both have developed alongside the advancement of networking technologies. From a connectivity perspective, the main categories of evolution of distributed computing can be classified are shown in figure 2.2 below where arrows indicate their evolving trend over time.

Qiu notes that enterprise applications have been substantially improved in terms of functionality & availability in serving end users needs, made possible as a direct result of evolutions in connectivity coupled with the general evolution in programming languages from sequential, to procedural & eventually to Object Oriented languages.

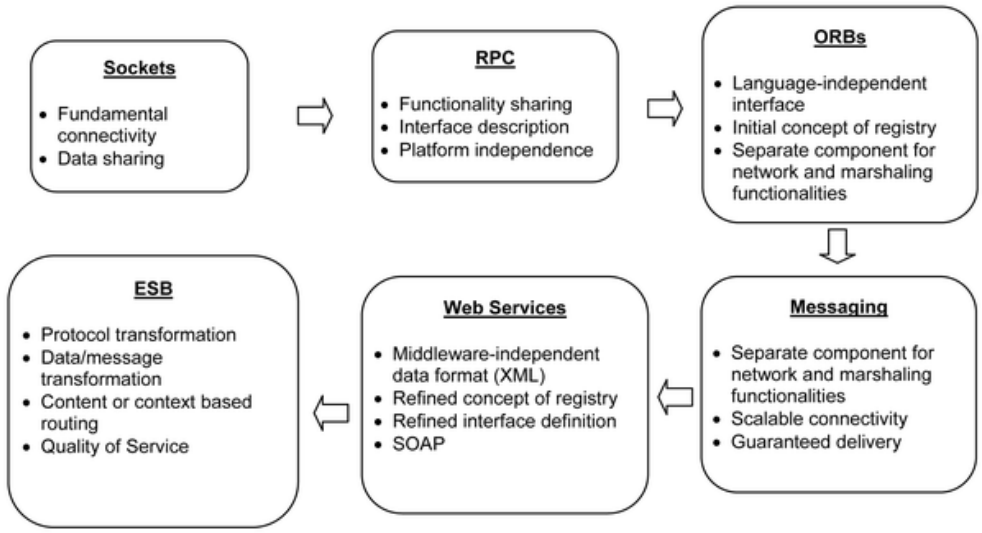


Figure 2.2 The evolution of distributed computing: the connectivity perspective [Qiu, 2013]

Additionally, the programming models sequentially applied to deliver distributed applications is another key enabler in the shift to distributed computing. It is beyond the scope of this dissertation to consider each of these models in great detail but they are useful to review for comparative purposes & as such, are briefly described below leaning heavily on Qui's (2013) summary.

2.1.1.1 Remote Procedure Call (RPC) Programming Model

The RPC programming model is a first generation approach at sharing computing services between distributed applications. The model typically relies on the request-reply communication protocol resulting in a fully-coupled communication between applications. Underlying RPC support also varies with the programming language & therefore not amenable to heterogeneous computing.

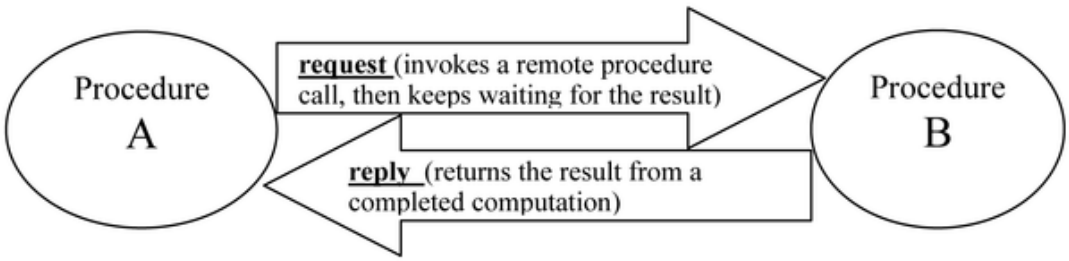


Figure 2.3 RPC Programming model [Qiu, 2013]

2.1.1.2 Object-Oriented (OO) Programming Model

The Object-Oriented programming model (Meyer, 2000) emerged as a result of the birth of OO programming languages & usually has no significant difference from RPC programming model i.e. still uses the request-reply communication protocol & frequently results in a fully-coupled communication between applications. The methods however are grouped together using class definitions & not all methods in a given class are remotely accessible. Similar to RPC, there are several different programming languages that support OO development with varying support for the necessary underlying remote object invocation mechanism.

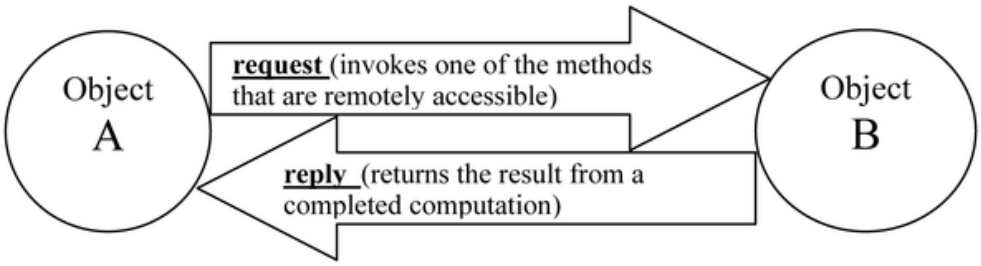


Figure 2.4 Object-oriented Programming model [Qiu, 2013]

2.1.1.3 Interface Based Programming Model

Interfaces are basically considered as abstraction layers in programming. This model takes advantage of the emergence of the concept & corresponding programming technologies to improve the interoperability, robustness & maintainability of enterprise applications. The change from OO programming to interface-based programming is not substantial but the addition of the abstraction layer improves interoperability by encapsulating the implementation details.

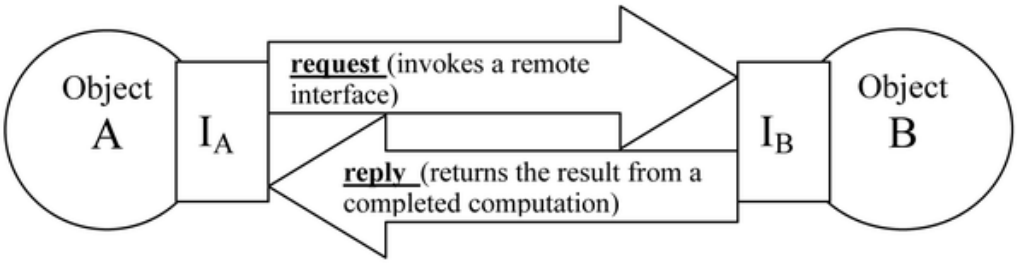


Figure 2.5 Interface-based Programming model [Qiu, 2013]

2.1.1.4 Component-based programming model

The component-based programming model (Heineman & Councill, 2001) made possible the development of software applications by assembling software modules, often built by other providers, when a preset specification is commonly compiled by such providers. This greatly improves reliability within an enterprise in cases. For example, where one service component becomes instable for some reason & a different service is used as a substitute by a solution provider, which delivers more reliably. It also allows for reusability of components internally within an enterprise where multiple internally delivered applications require the same type of component level functionality.

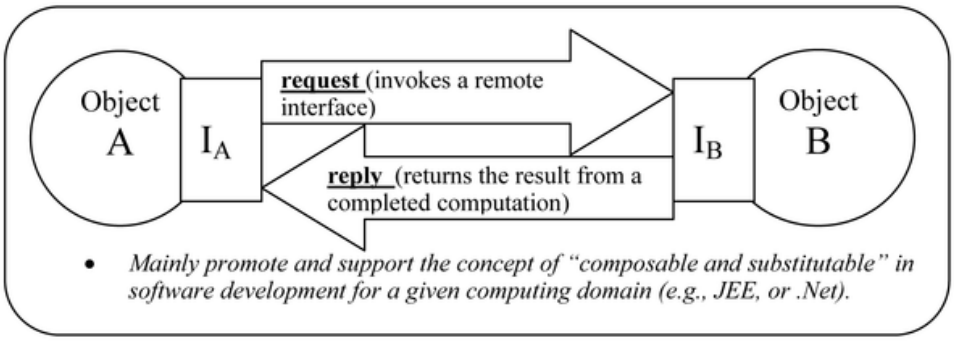


Figure 2.6 Component-based Programming model [Qiu, 2013]

2.1.1.5 Service-based programming model

The computing model known as 'Service-Oriented' allows enabled services to be programming-language, executing platform & integration / middleware independent. These characteristics are key to facilitating the interconnection of devices in a heterogeneous network. Qiu (2013) concludes this review by observing that from a business support perspective, a computing service is essentially encapsulated to represent a unit of business work. We explore the concept of 'web services' enabled by Service Oriented architectures in 2.1.3.1 below.

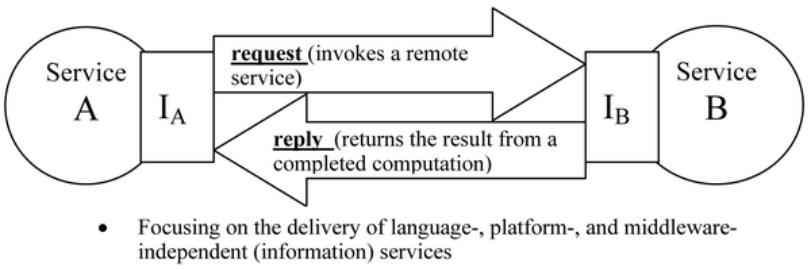


Figure 2.7 Service-oriented Programming model [Qiu, 2013]

2.1.2 Design Considerations in Distributed Systems

In any form of computing there are trade offs between it's advantages & disadvantages. Mahajan & Shah (2013) highlight the following key design issues relating to the development of distributed system. We review them here in the context of their criticality for the delivery of a highly distributed enterprise offering.

- **Transparency**: The most important design issue is to hide from users the fact that processes & resources are physically distributed across the network. There are several types of transparency.

Transparency	Description
Access	Hide the difference in data representation and how a resource is accessed
Location	Hide where a resource is physically located
Migration	Hide the movement of a resource to another location
Relocation	Hide the movement of a resource to another location while in use
Replication	Hide the fact that multiple copies of the resource exist without the user's knowledge
Concurrency	Hide the fact that a resource may be shared by several users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a resource is in memory or on disk

Figure 2.8 Types of transparencies [Mahajan, Shah 2013]

- **Flexibility:** Abstracted, this refers to the ability to make modifications to decisions made at initial distributed system design.
- **Reliability:** In distributed systems, multiple processors are available for computation. Where one processor crashes, a backup is available. Reliability in terms of data means that application data should be available at all times without error. In the case of replication, all copies of data should be consistent. Extra precautions are needed in systems that require frequent updates.
- **Performance:** Applications should run in a distributed environment just as if it were running on a single processor. Common metrics used to measure the performance of a distributed system include response time, throughput, system utilization & the amount of network capacity used. Message transmission over a

LAN takes some time, on the order of 1 millisecond. Performance optimization requires that we minimize the number of messages & hence network traffic. Two commonly used mechanisms to achieve this are caching data that is repeatedly needed & batching messages together.

- **Scalability:** Distributed systems are designed to work with hundreds of CPU's & are designed to expand over time with evolving requirements. Supporting more users or resources exposes limitations with centralized services, tables & algorithms. Traditionally centralized services are implemented by a single server running on a specific machine in the distributed system. The solution is to distribute services across multiple servers & prone to another set of issues summarized in the table below.

Concept	**Example**
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	A single algorithm doing routing based on available information

Figure 2.9 Scalability related issues [Mahajan,Shah 2013]

Techniques used to scale distributed systems include hiding communication latencies via asynchronous communication, hiding distribution e.g. Internet DNS & hiding replication by handling consistency problems.

- **Security:** In a distributed system, user information needs to be secure. This comprises 3 main aspects: confidentiality (protection against unauthorized access), integrity (protection of data against corruption) & availability (protection against failure & always being accessible). Generally information is encoded & sent across a network in a secure manner. Security also involves identifying the user who is sending the message & encryption is used to solve this issue.
- **Fault Tolerance:** Distributed systems are designed to mask failures. If one server breaks down, the other server takes over in a manner that is transparent to the user.

These design considerations are at the heart of any distributed computing model & any application being built to operate in a distributed computing environment. As

such, they are relevant & equally applicable to all of the distributed computing models explored in the next section.

2.1.3 Overview of Modern Distributed System Models

The general computing trend is to leverage shared web resources and massive amounts of data over the Internet. Figure 2.10 below illustrates the evolution of High-Performance Computing (HPC) and High-Throughput Computing (HTC) systems as described by Hwang, Fox & Dongarra (2103).

On the HPC side, supercomputers *(*massively parallel processors *or* MPPs*)* are gradually replaced by clusters of cooperative computers out of a desire to share computing resources. The cluster is often a collection of homogeneous compute nodes that are physically connected in close range to one another

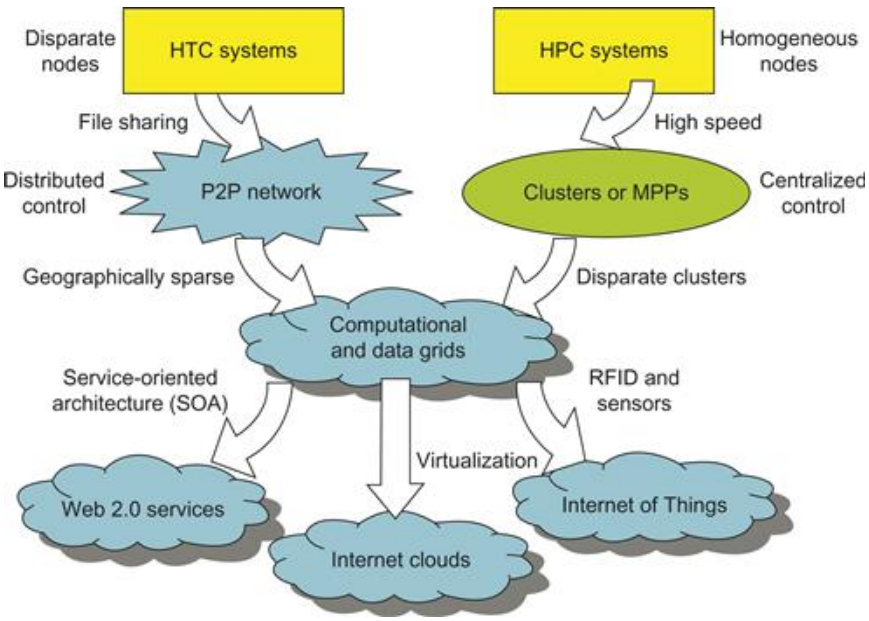


Figure 2.10 Evolutionary trend toward parallel, distributed, and cloud computing with clusters, MPPs, P2P networks, grids, clouds, web services, and the Internet of Things. [Hwang, Fox & Dongarra 2013]

On the HTC side, peer-to-peer (P2P) networks are formed for distributed file sharing and content delivery applications. A P2P system is built over many client machines. Peer machines are globally distributed in nature. P2P, cloud computing, and web service platforms are more focused on HTC applications than on HPC applications. Clustering and P2P technologies lead to the development of computational grids or data grids.

With the introduction of SOA discussed in 2.1.1.5 above, Web 2.0 services become available. Advances in virtualization make it possible to see the growth of Internet clouds as a new computing paradigm. The maturity of radio-frequency identification (RFID), Global Positioning System (GPS), and sensor technologies has triggered the development of the Internet of Things (IoT). These new distributed computing paradigms have been the subject of much hype & new academic as well as industry literature is starting to emerge. We will briefly review each here before looking at Cloud Computing in more detail.

2.1.3.1 Web 2.0 Services

In an SOA paradigm, software capabilities are delivered and consumed via loosely coupled, reusable, coarse-grained, discoverable, and self-contained services interacting via a message-based communication model. The term "web service" is often referred to as a self-contained, self-describing, modular application designed to be used and accessible by other software applications across the web (Hwang, Fox & Dongarra, 2103). Once a web service is deployed, other applications and other web services can discover and invoke the deployed service.

A web service is one of the most common instances of an SOA implementation. The W3C working group defines a web service as 'a software system designed to support interoperable machine-to-machine interaction over a network' (Booth, Haas, McCabe et al, 2004, p.7). According to this definition, a web service has an interface described in a machine-executable format (specifically Web Services Description Language or WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards.

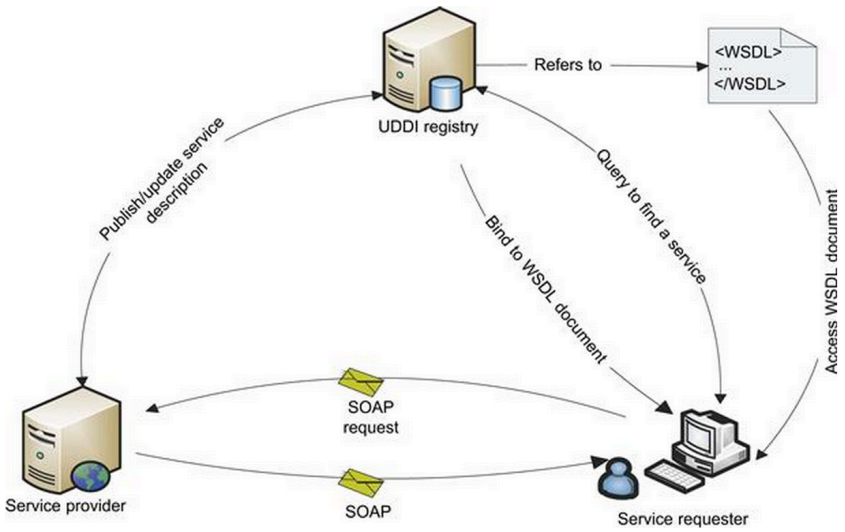


Figure 2.11 A simple web service interaction among provider, user, and the UDDI registry - [Hwang, Fox & Dongarra 2103]

RESTful web services, the subject of this research can be considered an alternative to SOAP stack or "big web services," because of their simplicity, lightweight nature, and integration with HTTP. With the help of URIs and hyperlinks, REST has shown that it is possible to discover web resources without an approach based on registration to a centralized repository. Section 2.4 looks at the detail of how REST achieves this in more detail.

2.1.3.2 The Internet of Things

"With a trillion sensors embedded in the environment—all connected by computing systems, software, and services—it will be possible to hear the heartbeat of the Earth, impacting human interaction with the globe as profoundly as the Internet has revolutionized communication."

- PETER HARTWELL, SENIOR RESEARCHER, HP LABS

According to the Cisco Internet Business Solutions Group (IBSG), IoT is simply 'the point in time when more "things or objects" were connected to the Internet than people' (Evans, 2011, pg.2).

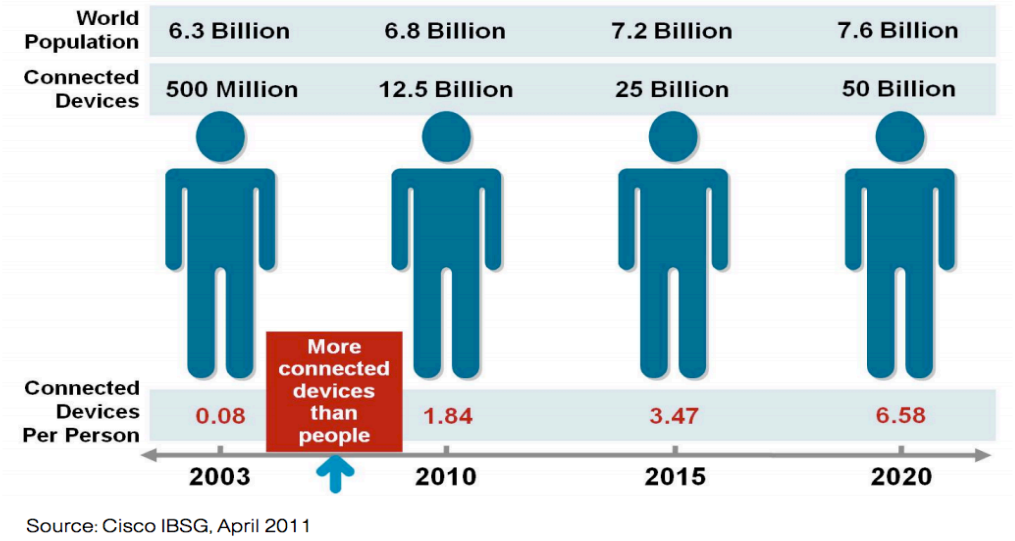


Figure 2.12 The Internet of Things Was "Born" Between 2008 and 2009 [Cisco IBSG, 2011]

In 2003, there were approximately 6.3 billion people living on the planet and 500 million devices connected to the Internet. By dividing the number of connected devices by the world population, the IBSG found that there was less than one (0.08) device for every person. Based on this definition, IoT didn't yet exist in 2003 because the number of connected things was relatively small given that ubiquitous devices such as smartphones were just being introduced. For example, Steve Jobs, Apple's CEO, didn't unveil the iPhone until January 9, 2007 at the Macworld conference.

Looking to the future, Cisco IBSG predicts there will be 25 billion devices connected to the Internet by 2015 and 50 billion by 2020. The magnitude of this projection in terms of the business opportunity it represents & the potential applications of the technology are described in detail by Evans (2011) in the Cisco whitepaper on how the next evolution of the Internet is changing everything.

While the industry is working on overcoming the some short-term barriers to the proliferation of IoT (Coetzee & Eksteen, 2011) the criticality of exposing new & existing services for emerging 'things' to connect to via industry standard Web 2.0 services increases. Organizations that do not are at risk of being left behind on the next wave of the Internet evolution.

2.2 Cloud Computing

"There was a time when every household, town, farm or village had it's own water well. Today, shared public utilities give us access to clean water by simply turning on the tap; cloud computing works in a similar fashion. Just like water from the tap in your kitchen, cloud computing services can be turned on or off as quickly as needed. Like at the water company, there is a team of dedicated professionals making sure the service provided is safe, secure & available on a 24/7 basis. When the tap isn't on, not only are you saving water, but you aren't paying for resources you don't currently need."

- VIVEK KUNDRA, FORMER FEDERAL CIO, U.S. GOVERNMENT

Cloud Computing is a tsunami of transformation amassing immense wealth for companies that didn't exist a few years ago – such as Google, Facebook, Amazon, Salesforce.com & Zynga; disrupting long-standing business models & ecosystems including publishing, advertising, television, the recording industry, telecommunications & retailing; and reordering relationships within the computing industry; among hardware vendors, licensed software vendors, distributors, value-added resellers, & systems integrators, to name but a few.

2.2.1 The Business Case for Cloud Computing

In both consumer & enterprise businesses, the cloud represents an existential threat & an irresistible opportunity. According to Weinman (2012), virtually any summary of key trends or chief information officer (CIO) focus areas ranks cloud computing at or near the top of the list. A Gartner survey in 2011 of 2000 CIO's around the time of his publication, an eternity in the computer industry, placed cloud computing as the number-one technology priority (Gartner, 2011).

Most, if not all of the rest of the top priorities – virtualization, mobility, collaboration, & business intelligence – enable, or are enabled by, or otherwise relate to the cloud. 3 years later, the situation has not changed significantly other than to indicate that cloud computing continues to be an enabler for the other trends that start to emerge such as the Internet of Things reviewed above (Gartner, 2013 & High 2013).

So the cloud is the new thing, but what it actually is & how it is defined in the literature reviewed is very much dependent on the author & what they expect from cloud computing (Amanatullah, Lim, Ipung & Juliandri, 2013).

Weinman (2012) coined the term 'Cloudonomics' in the summer of 2008 for Cloudnomics.com & a blog post for the popular technology site GigaOM.com (Weinman, 2008b), syndicated to BusinessWeek (Weinman, 2008a). It is in this context that his book defines the cloud with a helpful mnemonic, C.L.O.U.D., reflecting five salient characteristics:

- **C**ommon infrastructure
- **L**ocation independence
- **O**nline accessibility
- **U**tility pricing
- **O**n-Demand resources

It is *common*, in that it uses pooled resources & dynamically shared infrastructure; *location-independent*, in that the service should be ubiquitous & responsive; *online*, that is, accessed over a network; a *utility*, creating value & with usage-sensitive pricing; and *on-demand*, that is, with the right quantity of resources available exactly & only when needed. Weinman claims that such a perspective is one of the most important ways to evaluate & exploit the cloud, since '*unless a technology drives compelling value, it will end up in the dustbin of history'*.

Kavis (2014, pg.8-10), presents a number of compelling real-life examples of the business savings realized by his own start-up experience by leveraging cloud computing resources that follow a 'pay-as-you-go' pricing model just like electricity & water. He also validates Weinman's claims above by looking at four different case studies across four different sectors to leave no question in the minds of the reader as to either the applicability of cloud computing or the 'compelling value' that it provides.

- **Start-Up** Case Study: Instagram, from Zero to a Billion Overnight
- **Established Company** Case Study: Netflix, Shifting from On-Premises to the Cloud

- **Government** Case Study: NOAA, E-Mail, & Collaboration in the Cloud
- **Not-for-Profit** Case Study: Obama Campaign, Six-Month Shelf-Life with One Big Peak

The bulk of the literature takes a more technology-focused mindset targeting cloud architects & gets specific about how to build software in the cloud, focusing on certain aspects of the technology (Mollah, Islam & Islam, 2012). It is beyond the scope of this dissertation to provide a fully comprehensive framework for building cloud software but we will review the base cloud computing layers & service models in order to help the reader understand the role REST API's play in the context of todays cloud computing environments.

2.2.2 Cloud Computing Layers & Service Models

A cloud is presented to the user as a computing environment, accessed through the Internet, with apparent infinite hardware resources. Clouds are large data centers owned by specific companies who make them available through pay-as-you-go schemes described above. Most of them use the concept of virtualization for both giving access to the user & offering flexible hardware configurations (Escalera & Chavez, 2012).

Some of today's most popular public clouds include Google's AppEngine, Amazon's AWS, IBM's Blue cloud & Microsoft's Azure. Each provider organizes its cloud in a different way but there is a certain basic architecture composed of layers in which they fit as Kavis (2014) effectively illustrates below.

There are three cloud service models: Software as a Service (SaaS), Platform as a service (PaaS), and Infrastructure as a Service (IaaS). Each cloud service model provides a level of abstraction that reduces the efforts required by the service consumer to build & deploy systems.

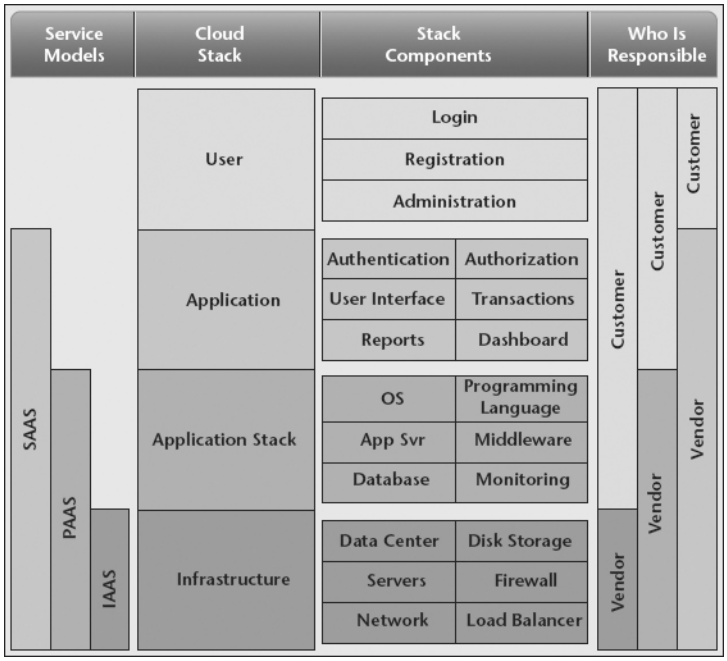


Figure 2.13 Cloud Stack [Kavis, 2014]

2.2.2.1 Infrastructure as a Service (IaaS)

The National Institute of Standards & Technology (NIST) defines IaaS as "the capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deply & run arbitrary software, which can include operating systems & applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage & deployed applications & possibly limited control of select networking components e.g. host firewalls" (Mell & Grance, 2011).

There are several IaaS vendors in the marketplace, the most mature & widely used being Amazon (AWS). However, Rackspace is another key player & OpenStack is an open source project that provides IaaS capabilities for those consumers who want to avoid vendor lock-in & want the control to build their own IaaS capabilities in house, referred to as private cloud. The challenges associated with lock-in are effectively described by Escalera & Chavez (2012).

2.2.2.2 Platform as a Service (PaaS)

The next layer up is PaaS. What IaaS is to infrastructure, PaaS is to applications. PaaS sits on top of IaaS & abstracts much of the application stack-level functions & provides those functions as a service.

The Cloud Security Alliance (CSA), a standards organization for cloud security describes PaaS as "the delivery of a computing platform & solution stack as a service. PaaS offerings facilitate deployment of applications without the cost & complexity of buying & managing the underlying hardware & software & provisioning hosting capabilities" (CSA, 2011).

2.2.2.3 Software as a Service (SaaS)

At the top of the stack is SaaS – the complete application delivered as a service to the service consumer. The service consumer has only to configure some application-specific parameters & manage users. The service provide handles all of the infrastructure, all of the application logic, all deployments & everything pertaining to the delivery of the product or services. This has very real implications for the next generation of software developers in their role & responsibility for the delivery of such services & guaranteeing up time.

Companies chose to rely on SaaS solutions for non-core-competency functions so they do not have to support the application infrastructure, provide maintenance & hire staff to manage it all. Instead they pay a subscription fee & simply use the service over the Internet as a browser-based service.

2.2.2.4 Deployment Models

NIST defines four key deployment models depicted below in their version of the definition of Cloud Computing.

- **Public Cloud:** The cloud infrastructure is provisioned for use by the general public. It may be owned, managed & operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider

- **Private Cloud:** The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g. business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, & it may exist off premises.

- **Hybrid Cloud:** A composition of two or more distinct cloud infrastructure (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

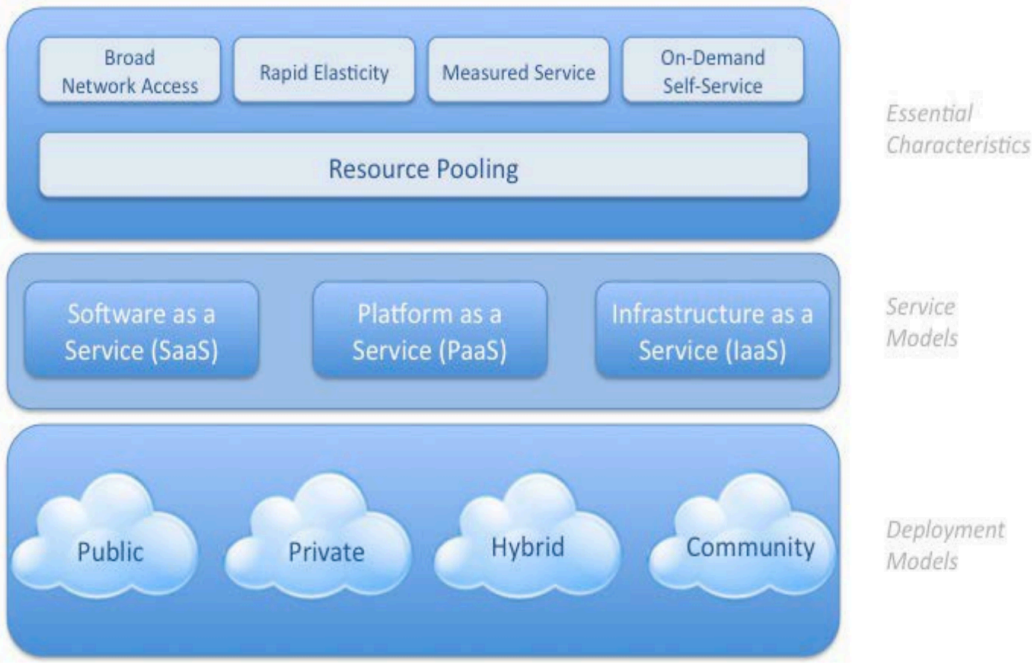


Figure 2.14 The NIST Definition of Cloud Computing

Choosing the right service & deployment model is a critical factor for delivering cloud based solutions & many frameworks exist to support this. Kavis (2014, p. 55-70) presents a practical case study & criteria for selection are becoming well published in the industry. However, more central to the subject of this research is that figure 2.14 depicts cloud computing as a set of abstraction layers, each providing a different service by leveraging a set of core API's.

2.2.3 REST as it relates to Cloud Computing

There are many reasons why REST based services are a critical component of any cloud-hosted solution, we will consider three of them.

When building services to be delivered from the cloud, one typically leverages one or more of the three layers of cloud computing introduced above. All of the major cloud

service providers expose their API's using RESTful services including Amazon, Rackspace etc.

By leveraging API's at the IaaS & PaaS layer to access numerous third party solutions, developers can provide fail over, high service level agreements (SLAs), & achieve huge gains in speed to market & cost efficiency since they don't have to manage & maintain the technology behind the API's. At the SaaS layer integrations with other REST based API providers is also common place e.g. how quickly we can plug-in to Facebook or Twitter API's. The fact that these leading vendors at each layer of the stack have already adopted REST implies at least some level of existing expertise using this style of development within the cloud computing community. Leveraging that expertise is a sensible approach for any new cloud hosted service.

A second reason that RESTful services are a critical component of any cloud solution pertains to the many touch points that users consume information on. Developer productivity is central to efficiency in start ups as well as large enterprise & cloud centric continuous delivery models abound (Humble, 2014). Building services once that are consumed across multiple devices & platforms rendering UI & keeps the apps in sync is a generally accepted architecture & supports such productivity.

The last & most important reason is that in cloud computing, every piece of infrastructure is expected to fail. The cloud is designed to be fault tolerant. For software to achieve fault tolerance it cannot be tied tightly coupled to the infrastructure on which is runs. This is a core principle for building REST web services & will be explored in detail in the next section.

2.3 Representation State Transfer

Representational State Transfer (REST) is a software architecture style for distributed systems, particularly distributed hypermedia systems, such as the World Wide Web. It has gained popularity among enterprises such as Google, Amazon, Yahoo!, and especially social networks such as Facebook and Twitter because of its simplicity, and its ease of being published and consumed by clients. REST, summarized in Figure 2.16 below, was introduced by Fielding (2000), one of the principal authors of the

HTTP specification, in his doctoral dissertation & was developed in parallel with the HTTP/1.1 protocol.

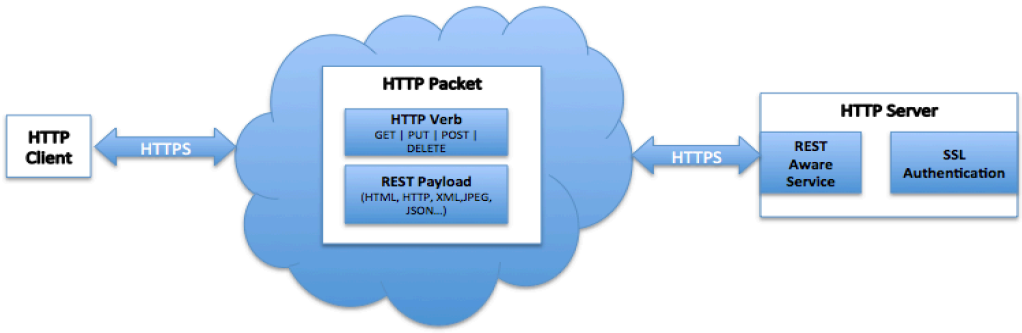


Figure 2.15 REST Architectural Style

Fielding (2000) described the software engineering principles guiding REST & the interaction constraints chosen to retain those principles. He looked at how the Internet, a highly distributed set of independent resources, work collectively with no knowledge of any resource located on any server & generalized it to apply the same concepts to REST by asserting four major constraints described below. Lee & Chou (2011) effectively summarize the implications of failing to satisfy each constraint during their development of a Petri Net approach to design & describe a REST API without violating REST. The summary below is annotated with their perspectives on this.

2.3.1 Separation of resource from representation

The key abstraction of information in REST is a resource. A resource is described as 'any information that can be named', any concept that might be the target of the author's hypertext reference' and 'a conceptual mapping to a set of entities' *(Fielding 2000, p. 88)*. Where the latter definition is concerned, he continues to explain that the values in the set may be resource representations and/or resource identifiers. REST uses a resource identifier to identify the particular resource involved in an interaction between components.

A somewhat simpler description of a resource is *'a single instance of an object'* (Hunter II, 2013). Some resources by definition are static (value set will rarely change

after creation), others have a high degree of variance in their value over time. The only thing that is required to be static for a resource is the semantics of the mapping.

A representation is used to capture the current or intended state of a resource & for transferring that state between components. It typically consists of data & metadata describing the data in the form of key / value pairs. The data format of a representation is known as a media type. Today's predominantly used media type used by RESTful interfaces is JSON (JavaScript Object Notation) but XML, HTML, SOAP or any valid data format could legally be used.

In essence, this constraint mandates that resources & representations must be loosely coupled e.g. a resource may be a data store or chunk of code, while the representation might be an XML or JSON result set or an HTML page. Failure to satisfy this constraint implies that clients are assuming a resource structure due to out of band information, such as a domain-specific standard, which is the data oriented equivalent to RPC's functional coupling (Lee & Chou, 2011).

By separating the resource from the representation, it becomes possible to scale the different components of a service independently, examples of which are described by Kavis (2014, pg. 73).

2.3.2 Manipulation of resources by representations

This constraint mandates that a representation of a resource with any metadata attached provide sufficient information to modify or delete the resource on the server, provided the client has permission to do so.

This basically says that a resources data (some row in a database), can only be modified or deleted on the server if the client sending the representation (a JSON or XML file) has permission to do so – permissions are determined by validating the user credentials embedded in the resource representations media type. Failure here implies again that out-of-band information is driving interaction instead of hypertext (Lee & Chou, 2011).

2.3.3 Self-descriptive messages

Each message provides enough information to describe how to process the message. In other words, messages must contain information that describes how to parse the

data. For example, a publically accessible REST API such as Twitter's has an unknown population of consumers therefore they support two different output formats for their services – XML & JSON. Consumers must describe in the request which format their incoming messages are so that Twitter knows which parser to use to read the messages. This supports maintaining a single version of the service by simply adding new parsers as needed. As with constraint 1, failure to satisfy this again implies that clients are assuming a resource structure due to out of band information (Lee & Chou, 2011).

2.3.4 Hypermedia as the engine of application state (HATEOAS)

This is arguably the most important yet most controversial of the constraints described. It is also one of the least likely to be followed as is evidenced by a blog post (Fielding, 2008) reiterating that REST API's need to be hypertext-driven before being classified as RESTful.

The constraint mandates that the client interacts with applications only through hypermedia (e.g. hyperlinks). It is how RESTful services work without maintaining application state on the server. By leveraging HATEOAS, the application state is in effect represented by a series of links – uniform resource identifiers or URI's – on the client side, similar to following the site map of a website by following URL's. When a resource fails, which cloud computing assume to be the rule, the resource that resumes working on the services starts with the URI of the failed resource (the application state) & resumes processing. The constraint therefore supports building in fault tolerance, which is critical to distributed, & cloud computing. Failure here implies again that out-of-band information is driving interaction instead of hypertext (Lee & Chou, 2011).

Hunter II (2013), notes that Hypermedia API's go back to the roots of how HTTP & HTML were intended to work. When working with non-Hypermedia RESTful API's, the URL endpoints are part of the contract between the server & the consumer. These endpoints must be known by the consumer ahead of time & any modifications break backward compatibility. The apparent issues inherent in this form of 'tight coupling' are well documented. Robinson (2006), provides some key arguments for avoidance of this type of tight coupling by way of practical example.

The Hypermedia API concept works the same way a human would surfing the web, clicking links that look interesting while generally exploring a website. The developer requests the root of the API which returns a list of URLs which point perhaps to collections of information & describing the collection in way that a human can understand. In this fashion, URLs are always up to date within responses & do not need to be known beforehand as part of a contract. When we GET a particular entry in a data store for example, we discover where we can go next, similar to the operation of a state machine as an application moves through transition states (Kavis, 2014).

If a URL is ever cached, & a subsequent request returns a 404, the consumer can just go back to the root & discover the content again. When retrieving a list of resources within a collection, an attribute containing a complete URL for the individual resource are returned which supports a 3xx redirect to the complete resource later upon a POST/PATCH/PUT request if necessary.

However, questions have been raised over the validity or pragmatism of this constraint (Knupp, 2014). Specifically the idea that simply embedding enough information in request responses that allow the client to 'intelligently' choose what action to take next using only the contents of the response. Critics point out that in our efforts to design Hypermedia API's, we are designing for clients that do not, & will never exist – that is, a smart client that could somehow make use of every single 'properly built' REST API in existence without requiring documentation. Knupp (2014) asks what that would even mean.

HATEOUS & the documented & undocumented challenges associated with it will be explored in more detail along with the other constraints outlined above by way of practical example in the succeeding chapters.

2.3.5 Why the four REST constraints are important for building cloud solutions

The cloud, just like the internet on which REST is based, is a massive network of independent resources that are designed to be fault tolerant. By following the constraints of REST first described by Fielding (2000), the software components that run in the cloud have no dependencies on the underlying infrastructure that may fail at any time. When not followed, it creates limitations on the application's ability to scale

& to fail over to the next available resource which, as noted in section 2.3, are the key premise on which cloud computing is based.

However, practical application of these principles have unearthed the trade offs inherent in all software architectures. The more abstraction that is built into the architecture, the more flexible & agile it becomes but at a cost of more up-front design investment & scope for performance degradation.

Abstraction creates overhead which can impact the latter & use cases exist where the performance requirements far exceed the benefits of adherence to a fully compliant RESTful API. Collaboration specific examples will be explored in Chapters 4 & 5 where the API designed & implemented is assessed for REST compliance against the Richardson Maturity model (Fowler, 2010).

Chapter Three

3. RESEARCH FRAMEWORK & METHODOLOGY

For the purposes of describing the research framework & methodology, the research hypothesis discussed in the introduction can be distilled as follows:

Real-time, cloud hosted applications that rely exclusively on the use of fully compliant RESTful API's, perform sub optimally to an alternate, hybrid approach within the same problem domain

The research objective was to validate the condensed hypothesis above by way of practical example described within the context of a broader, real world use case on a scale appropriate to the project. Based on the findings, present a set of recommendations that can be applied by others seeking to satisfy a similar set of requirements.

The steps involved in achieving that objective may be summarized as follows:

- The completion of a comprehensive literature review to fully understand the advancement of computing models toward the mobile & cloud computing waves (or 3rd platform) as well as the distributed computing design patterns integral to that evolution. Specifically, the foundations & rationale of the REpresentational State Transfer architectural style. Acclaimed textbooks & research papers dealing with distributed, cloud & mobile computing were studied & evaluated. This served as a solid theoretical basis for the development of a real world enterprise use case around which the rest of the research was based.

- The formal definition of a set of real world, enterprise focused collaboration requirements. These are presented in the form of industry standard, agile user stories & entered into the Rally Enterprise in order to represent Acme Inc.'s (a fictitious start up) product vision. This case study encompasses the business benefits of cloud & mobile compute as well as a series of functional

requirements that are intended to deliver the desired user experience.

- A cloud based architecture & delivery model is proposed in order to deliver Acme's product vision which will not be demonstrated by way of practical example as beyond the scope of this thesis, but does provide context for the overall solution & an understanding of the framework in which the REST endpoints implemented will operate. This was an important aspect to include for the case study analysis phase where performance evaluation is considered.

- The design & implementation of suite of server delivered, Tomcat hosted, Java based REST endpoints that offer a subset of Acme's cloud hosted collaboration services. These services were built to support a specific subset of Acme's complete suite of requirements in order to demonstrate the impact of two different approaches to the general design of real-time services.

- An evaluation of the API developed for compliance to the RESTful API constraints in the context of real time communication requirements such as Acme's using the Richardson Maturity model (Fowler, 2010).

- A performance evaluation of the Messaging implementation. This involved the development of a suite of JMeter tests that invoke the exposed REST endpoints, exerting load at a rate which is more analogous to the type of real world traffic a successful, cloud hosted, multi-device collaboration application would generate. The detail of the evaluation of each version is the subject of chapter 5. In keeping with the research hypothesis, version 1 was designed with the intent of being fully REST compliant, satisfying the constraints in Fielding's thesis. Version 2 deviates slightly to deliver the same functional requirements in what the hypothesis debates is a more optimal design.

Chapter Four

4. CASE STUDY DESIGN & IMPLEMENTATION

Industry experts agree that companies need their critical workforces to perform smarter, faster & more productively. Achieving that goal requires embedding collaboration technologies deep into processes & incentivizing collaborative behaviors – ultimately transforming the way organizations turn knowledge into action.

There is no lack of collaboration technologies in the marketplace as vendors continue to offer consumers & businesses a growing range of these tools. Facebook-like social platforms such as Chatter, Yammer & Jive, messaging platforms such as Cotap, WhatsApp & Snapchat as well as real time collaboration solutions such as Cisco WebEx & Jabber, Skype, Google Hangouts, Blue Jeans Video Conferencing, Pexip Infinity to name but a few.

Business leaders have a false sense of achievement however in their progress in this domain according to a global study by Avanade in late 2013. 74% of surveyed businesses are using Facebook to collaborate. When viewed in the context of their enterprise grade competitors that represents two times more than Microsoft SharePoint, four times more than IBM open connections, six times more than Salesforce Chatter (Avanade, 2013).

Consumer-driven technologies lack enterprise capabilities including document sharing, security, enterprise search & integration with real-time communication systems. The survey indicates that businesses are at the tipping point of a shift from consumer driven to enterprise class social collaboration tools in 2014.

Integrated chat messaging & real time meeting capabilities as part of this suite of emerging social collaboration tools will be fundamental to the success of this transition. At the same time, there is increasing evidence that organizations traditionally perceived to be consumer driven are making a strong push to enter the

enterprise collaboration market – a case in point being the launch of Business Hangouts that brings Google Hangouts to the enterprise for webinars, conferences, virtual classrooms & events etc. (Business Hangouts, 2014).

VidyoH2O is another example as the first product to result from Vidyo's collaboration with Google in early 2014. It lets companies use their existing videoconferencing hardware — whether it's from Cisco, Polycom, Avaya or other, with Hangouts. It also allows users to dial into Hangouts from regular phone lines, so remote users won't even need an Internet connection to join a conference, albeit in audio-only form (Humble, 2014).

4.1 Introducing Acme Collaboration Inc.

For the purposes of the business centric case study that follows, we introduce a fictitious company called Acme Collaboration Inc. The newly formed company has just secured funding from two venture capitalists in Silicon Valley & is planning to enter the collaboration market in November 2014 with the launch of a suite of collaboration products which will be unveiled at one of the industry's key annual conferences. This provides only a 6-month development cycle in which to satisfy the following high-level system requirements.

The goal of the Acme architecture is to create a platform that delivers a set of messaging & meeting services on top of which they can build simple & effective mobile, web & desktop collaboration apps that are fully synchronized across client endpoints. Of equal importance is the exposure of an open API layer that allows channel partners, app store developers & affiliate network partners integrate seamlessly to Acme's collaboration platform & core workloads.

They want to achieve this by building an Acme PaaS which will provide all the infrastructure & application logic required for 3rd party integration with these services. Transactions through it's API layer equate to new revenue streams that are not available in traditional premise based architecture & Acme believe this will give them the competitive advantage required to gain market share fast. They have secured relationships with a number of partners that are planning to integrate the services offered by the platform at launch to their existing cloud hosted LOB applications in

the CRM (Customer Relationship Management), ERP (Enterprise Resource Planning) & financial markets.

Acme is committed to a service-oriented-architecture (SOA) to enable the maximum degree of flexibility in the system & wants to leverage the cloud to achieve infinite horizontal scale at a lower price point than it's premise based competitors, while delivering greater speed to market.

To satisfy partner & customer security requirements, the platform needs to support a hybrid deployment model. As time & feature velocity is of the essence, the platform needs to encompass standard dev ops tooling support for monitoring service usage, detecting service outages & automated alerts to the development team in such scenarios.

Acme's software development team have adopted an agile approach to delivery of their software in incremental phases & will spend sprint 0 performing the necessary architectural discovery steps before diving headfirst into the implementation. The architects set out to seek answers to six key questions prior to recommending an architecture that can satisfy the system requirements above: Why, Who, What, Where, When & How.

Why	➢ Create a collaboration services platform to generate new business ➢ Mobile & Web to attract customers, drive more traffic ➢ Open API to attract 3rd party developers & enable partner integrations
Who	➢ Acme needs this to build a collaboration business that can be a market leader. ➢ End Users need this to leverage enterprise grade collaboration services ➢ Data center operations need to build infrastructure & dev ops capability ➢ Partners need this to integrate collaboration capability into existing LOB applications ➢ Unknown 3rd Parties: App Store developers ➢ Investors need this to get ROI
What	➢ New Collaboration services (Messaging & Live Meeting) ➢ Connect to partner LOB applications

	➢ Must scale to handle random spikes in traffic ➢ Must protect, secure all traffic ➢ Strong security given system is exposed to 3rd parties
Where	➢ End Users can live in any country ➢ Third party developers can live in any country ➢ Partners are US based but their users live in any country
When	➢ Product launch with one live partner integration by Nov (6 month development window)
How	➢ Limited experience in cloud & SOA ➢ Operating model is new – 3rd party support requires investigation

The functional & non-functional system requirements derived from this exercise were captured in the agile tool called Rally under the Acme Collaboration workspace (see Appendix A for details of how to access & the full list of requirements complied).

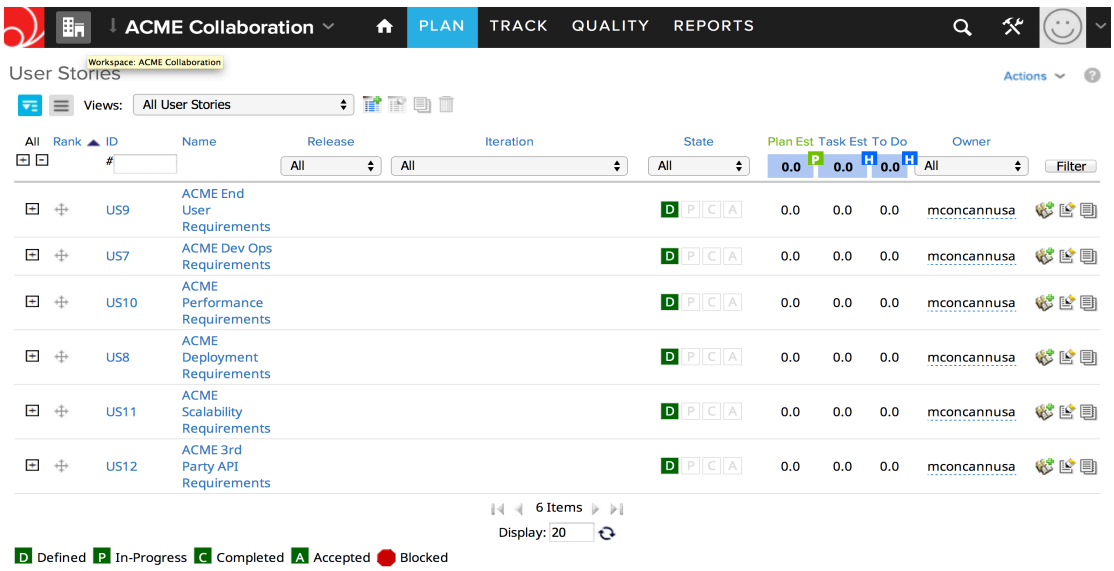


Figure 4.1 Acme User Stories as defined in the Industry Standard Rally Tool for Agile Development

4.2 Business Architecture & Constraints

A good first step when starting a major cloud initiative is to create a business architecture diagram. This is important because it provides insights into the various actors, touch points & business functions across the enterprise or at least across the part of the enterprise that is in scope for the initiative.

Figure 4.2 below describes Acme's business architecture & coupled with the system requirements detailed above, there are a series of constraints that the architects need

to consider before determining the right balance of cloud service & deployment models to meet these constraints.

1. **Technical constraints:** The architecture below highlights the fact that traffic loads will be dynamic in nature as third parties integrate & has the knock on effect that security is a significant factor.
2. **Financial constraints:** Reducing infrastructure costs is a key premise of the solution
3. **Strategic constraints:** Increase revenue via 3rd party integrations.
4. **Organizational constraints:** Lack of cloud & SOA skills
5. **Risk:** Slipping the six month deadline will represent missed market opportunity

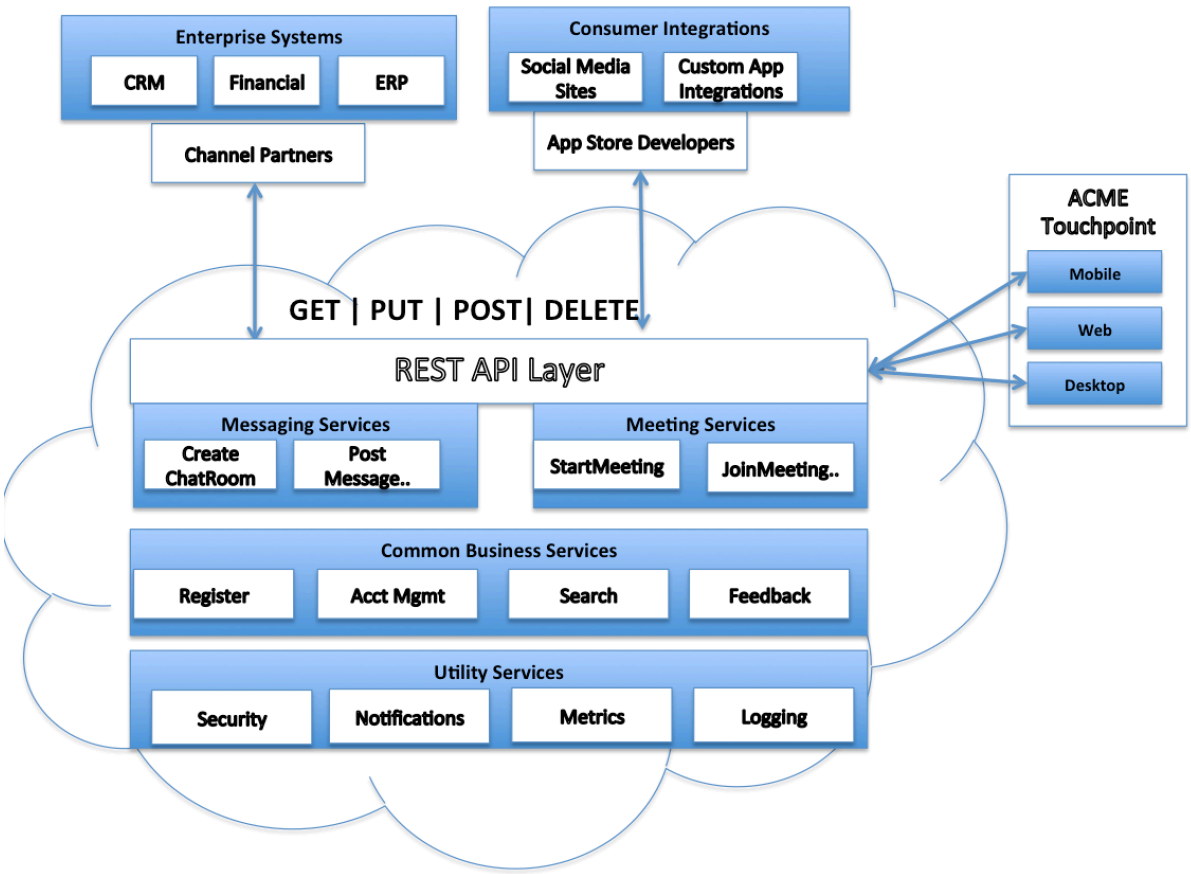


Figure 4.2 Acme Business Architecture

Time is a major constraint on Acme's plan so speed of development is paramount. The team at Acme lacks some of the cloud development skills necessary so to avoid introducing significant risk so there is obvious value in offloading anything that is not

a core competency. Section 4.4 considers the options at each layer in the cloud architecture to do just that.

4.3 Acme's Choice of Cloud Service & Deployment Model

As described in Chapter 3, choosing cloud service models & deployment models are critical tasks in any cloud computing initiative & decisions should be based on business drivers, constraints & customer impacts. Having now answered the six key architecture questions of Why, Who, Want, Where, When & How, drawing the high level business architecture & considering the system constraints inherent in the suite of Rally requirements, the next step is to consider options for leveraging SaaS, PaaS & IaaS solutions to accelerate speed to market, reduce risk & minimize infrastructure requirements. This exercise will allow the team at Acme to focus their design & implementation efforts on satisfying a subset of requirements in which they already have experience.

4.3.1 Where to use SaaS

A number of components in the high level business architecture described in figure 4.2 are candidates for leveraging SaaS based solutions:

- **API Layer.** It is feasible to leverage an API management SaaS tool to overcome the lack of experience writing Representation State Transfer (Restful APIs) in the cloud. Such tools can be used to provide support for multiple 3rd parties & manage API traffic etc. Thus providing an alternate approach to building from scratch the components necessary to satisfy some of the key. One such example used by Ticketfly to allow 3rd party resellers secure access to their backend servers is apigee. Apigee provides out of the box reliability, scalability & enterprise grade manageability for the API lifecycle (Apigee, 2014).
- **Feedback Reports.** A number of 3rd party systems exist that offer the ability to capture feedback gathered from the user during the course of using the application. One such example is Desk.com which provides a suite of API's that allow for integration with the saleforce.com CRM & Support offerings, removing the need to build an in house ticketing system to capture end user feedback (Salesforce, 2014)

- **Utility services.** All of the utility services are candidates for SaaS because they are not core competencies. However, it's plausible that they be provided by a PaaS or IaaS as well.

4.3.2 Where to use PaaS

The subset of components, which cannot be satisfied by SaaS solutions all require development. The next phase of analysis is determining which of those components could effectively leverage a PaaS service provider to get to market quickly without the need to manage infrastructure & the application stack. As the PaaS vendors have various limits in order to manage the performance, reliability & scalability of each customer, they enforce throttling once these limits are exceeded & thus, this is a significant factor to consider when choosing to proceed with any given PaaS. Designing around throttling can be expensive & may impact the quality, reliability & performance of the application.

Acme's two key cloud hosted services are messaging & meetings – both real time in nature & therefore any performance degradation would be imminently visible to a user. The projected usage pattern which will grow over time as more 3rd party developers leverage the platform as well as the associated performance requirements demand the ability to have control over the resources that could be provided by a PaaS (e.g. memory, database configuration, app servers to maximise throughput etc). For example, there is sufficient risk that in order to meet the user experience & system load requirements for the user stories below, direct IaaS access will be necessary.

As an *ACME Collab App User*
I want *any 1:1 real time meeting that I am invited to to be set up in under 1 second after I chose to accept the invite to it*
so that I *have the best experience possible*

As an *ACME Collab App User*
I want *any 1:1 notifications that I have generated as part of the messaging or real time meeting functionality to be received by the remote party in under 1 second*
so that *latency is minimised & we both have the best experience possible*

As an *ACME Collab App User*
I want *any message I post to a chat room to be instantly visible to me within the chat room*
so that I *have the best experience possible & do not experience latency while it is being centrally stored*

As an *ACME Collab Product Manager*
I want *the messaging & meeting services to support dynamic traffic loads based on varying usage*
so that *no upper limit ever needs to be set on the volume of simultaneous live chat rooms or meetings*

As an *ACME Collab App DevOps Prime*
I want *to understand the maximum system throughput of any single service instance deployed to production*
so that I *undestand the point at which I need to turn up another instance to avoid performance degradation or system outage*

However, Acme projects that it's messaging services will account for the bulk of the load in the first year & as such, rule out the use of PaaS for these services.

Real-time meeting traffic on the other hand is expected to grow steadily year over year & an iterative approach to delivering a fully IaaS based solution is feasible. Leveraging a PaaS provider for the first year of launching it's meeting services is a pragmatic approach to balancing the short-term need of getting to market quickly, against the long-term goal of scaling to Netflix levels. Cloud Foundry is a good choice of PaaS offering as an open source project that can be deployed on any infrastructure, reducing the risk of needing to move to another platform in the event that an alternate commercial vendor goes out of business (Cloud Foundry, 2014).

Other component candidates from the business architecture for a PaaS solution include:

Mobile Touchpoint: The team has little mobile development experience & a mobile development platform would accelerate the development process & reduce the amount of overall development.

Utility Services: The PaaS likely provides services for security, notifications, metrics & logging as well as support for some of the non functional requirements around alerting copied below from the suite of Rally requirements. Worth considering however is that the Messaging services running against an IaaS vendor will, by default leverage the utility services of that platform. One of two follow up steps is thus required if proceeding in this direction. Determining compatibility of the IaaS & PaaS utility services & proceed to use both or check if it's feasible to use a single set of utility services

As an *ACME Collab App Service Developer*
I want *access to a cloud hosted set of backend services required to build the Messaging & Meeting services for ACME.com*
so that I *can test & deploy the services in an environment that simulates production working with platform primes to provision services to meet the services resource requirements*

As an *ACME Collaboration DevOps Engineer*
I want *to know as much as I can about how the messaging & meeting services are being used by consumers of the service*
so that I *can make business decisions around what type of feature support to enable or deprecate*

As an *ACME Collaboration DevOps Engineer*
I want to know as much as I can about the success rate of requests to the messaging & meeting services in use by consumers of the service
so that I can investigate & troubleshoot issues in realtime & get a better understanding of the pain points in my architecture

As an *ACME Collaboration DevOps Engineer*
I want an effective set of logs for a configurable period generated from my services to be stored centrally with effective tool based search capability

so that I can quickly & effectively investigate & troubleshoot issues in realtime to recover my service as the situation demands it

***As an** ACME Collaboration DevOps Engineer*
I want to an automated alert to be sent to my phone if any of the deployed production services go offline
so that I can respond immediately to recover the service & minimize downtime for users of the service

4.3.3 Where to use IaaS

As noted above, where an application has significant performance / scalability requirements that require developers to manage memory, configure database servers & application servers to maximize throughput, specify how data is distributed across disks, manipulate the OS etc, it becomes necessary to leverage IaaS. While not defined in the user story requirements, the next iteration of detail like any large industry is likely to include key performance indicators such as delivering 1 million messages per minute or thousands of simultaneous meetings in the longer term. To achieve this Acme cannot be throttled by a cloud vendor & significant tweaking of the OS, database & application servers is likely going to be necessary. A PaaS model reduces the cost substantially by reducing the amount of work & resources required but gets expensive when data gets into terabytes or when bandwidth or CPU demands exceed 'normal' levels.

For the reasons cited above, messaging is considered to be the service that demands the highest throughput in this period & as such, is best suited to leverage IaaS directly. Another key advantage of going this route is to mitigate against downtime at the SaaS & PaaS provider layers. At each of those layers the customer can only wait for the provider to fix the issue & get services back online. Operating at the IaaS layer, Acme can architect for failure and build redundant services across multiple physical or virtual data centers.

Other component candidates from the business architecture for an IaaS solution include:
Utility services: As explored above, leverage the IaaS provided utility services.

Common business services: High volume services that are shared by both upstream messaging & meeting services.

The options & proposed service models above, illustrate the point that moving up through the cloud services stack toward SaaS increases the speed to market, reduces the number of developers required & reduces operational costs. Moving down the stack toward IaaS offers more control over the infrastructure & provides a better chance of avoiding or recovering from a vendor outage. This example illustrates how Acme can apply decisions at each layer to find the correct balance between speed & control in order to meet their specific near term priorities that will facilitate an on time delivery.

4.3.4 Acme Cloud Deployment Model

The final decision point before proceeding to design & implementation of the services being fully developed by Acme is that of which cloud deployment model makes sense given the business case, time constraints, organizational readiness & industry knowledge. To summarize the pertinent decision making factors for Acme's specific use case:

- There is a significant amount of personal identifiable information that is being passed through the system including messages that may be sensitive & Call Detail Records that fit the same category.
- Users & Partners may be located outside of the US & have concerns with data in the public cloud
- There is a risk of public PaaS & IaaS outages
- Requirement to reduce infrastructure footprint
- Need to get to market fast!

The public cloud is attractive where speed to market is concerned but there are compliance factors to consider where large enterprise customers are concerned. The public cloud may also scare aware security conscious & International 3rd parties. Leveraging a public IaaS provider however it is feasible to maintain uptime during PaaS outages with some investment in redundancy & based on the decision to phase migration of Meetings onto the IaaS platform, this investment is limited to that service. In the event of downtime, the highest priority service can be maintained.

Long term, a hybrid cloud solution makes the most sense. This would provide options for both the security conscious customer that wants all data to be maintained on premise (such as international partners) & can avail of the cloud for spikes in traffic. The public & private clouds can provide failover for each other as well as supporting a pure public offering for free, viral integrations with limited support. Leveraging a hybrid PaaS, Acme's solution can be guaranteed to run on both the public & private cloud.

However, the short-term aggressive deadline means that it will be challenging to invest in a private cloud solution & it has the disadvantage of deviating from the requirement to reduce infrastructure footprint. Therefore, the recommended path is to build a plan to progress toward the long-term goal of a hybrid deployment by starting with a public, cloud-only offering in the first year. The public cloud solution will need to incorporate redundancy across virtual data centers. Depending on the customer uptake in year one, it will be easier to justify the addition of incremental servers for the private cloud as users get excited about the public offering & expand their rollouts to levels that mandate the hybrid solution. This approach will also maximize the teams ability to meet the all important time to market requirements to avoid the missed business opportunity.

4.4 REST Implementation Scope

To prove the hypothesis described in Chapter 3, the code examples that follow have been built to deliver the subset of Acme User Stories copied from Rally below. These are used to demonstrate the role that REST plays in delivering cloud ready applications which is now the focus of Acme's development team given the decisions made above on what aspects of the overall solution can be delivered by cloud providers at the different layers.

User Stories 14, 15 & 18 which encompass some of the time sensitive aspects of Acme's messages service; posting a new message, being notified of a new message & read receipt delivery.

As an *ACME Collab App User*
I want *to be able to post a message in an existing persistent 1:1 chat room*
so that *it becomes visible to the other user in the chatroom asynchronously*

As an *ACME Collab App User*
I want *to receive a notification when a new message is posted to a persistent chat room that I am a member of*
so that *I become aware when the other participant is trying to communicate with me & I can chose whether to reply or not*

As an *ACME Collab App User*
I want *to know when the other person in the Chat Room has read the last message posted by me*
so that I *have some expectation of when they are likely to reply or if they are ignoring me / busy etc.*

For the purposes of the evaluation exercise that follows in chapter 5, the User Stories US35 & US38 (below) will also be incorporated.

As an *ACME Collab App DevOps Prime*
I want *to understand the maximum system throughput of any single service instance deployed to production*
so that I *understand the point at which I need to turn up another instance to avoid performance degradation or system outage*

As an *ACME Collab App Client Developer (Web, Mobile, Desktop)*
I want *a consistent & performant REST based API that can satisfy system requirements for Messaging & Meeting end user functionality*
so that I *am abstracted from as much system complexity as possible & can quickly & easily understand the resource model without much need for documentation or cross team design knowledge*

4.5 The Acme Messaging Service Implementation

The objective of the messaging implementation described below was three fold:

1. Design, build & test a sufficient set of 'REST compliant' endpoints that would form the basis of the concrete API evaluation & analysis that follows in chapter 5.
2. Validate the endpoints exposed could be consumed by some form of test client to prove / disprove the hypothesis outlined in chapter 2. It is not within the scope of this implementation to deliver an actual user interface based client, nor is it necessary to validate the thesis hypothesis.
3. Formulate a subjective opinion on an effective balance of REST conformance & real world pragmatism to apply in the context of a concrete collaboration example after completing the hands on implementation exercise. This objective was motivated by observing the challenge first hand as the leader of a team of very experienced & talented software engineers who make these trade offs daily.

Source code, JavaDoc, call flows, JMeter tests & result data can all be accessed at the publically accessible Git repository below but a high level summary of the design is included here. https://github.com/mconcanndev/msc_thesis

The Java based implementation was designed around 3 main resources illustrated below. In keeping with the REST approach to API design, each resource is addressable by it's own unique URI & the JSON body of each HTTP request / response operation contains the complete representation of the resource created or modified. The main entry point for the API is at /chatrooms where, HATEOAS compliance will guarantee that the URI's for the next accessible action are returned in the JSON response. From here, the consumer will navigate the relationship between ChatRoom & ChatMessage as well as User resources which has it's own independent entry point at /users.

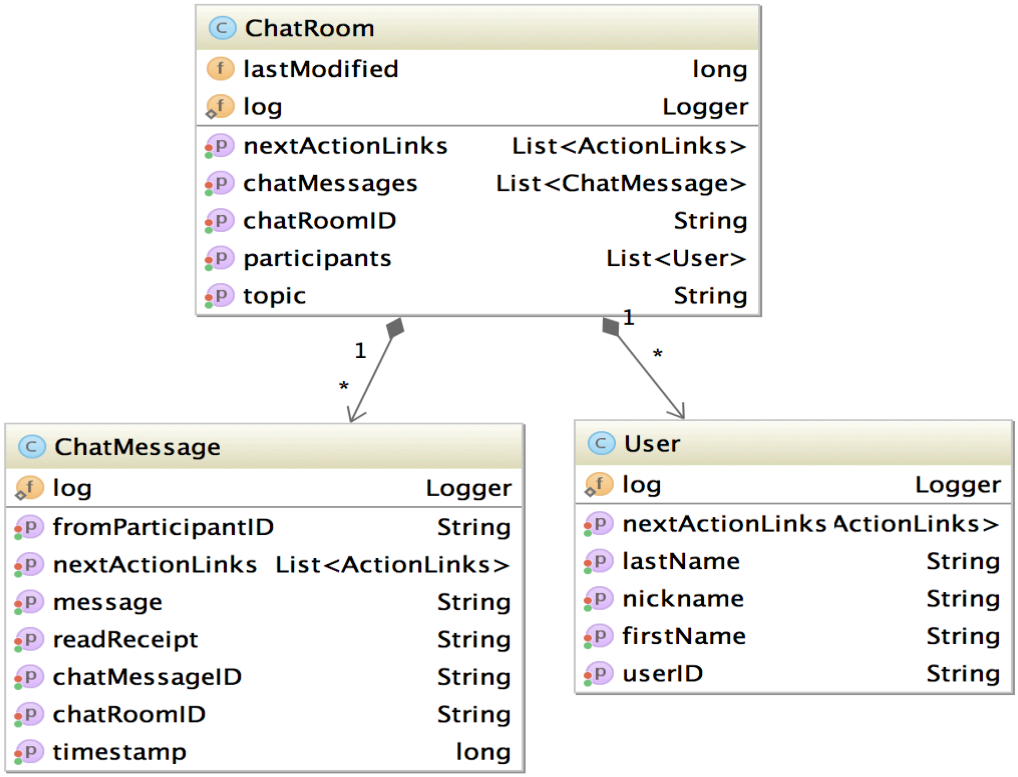


Figure 4.3 Key Resources around which the Messaging API implementation is designed

There are four primary layers in the overall design of the service: Controller layer, Service layer, DAO layer & Database layer. The general flow for any given REST endpoint invoked is illustrated below:

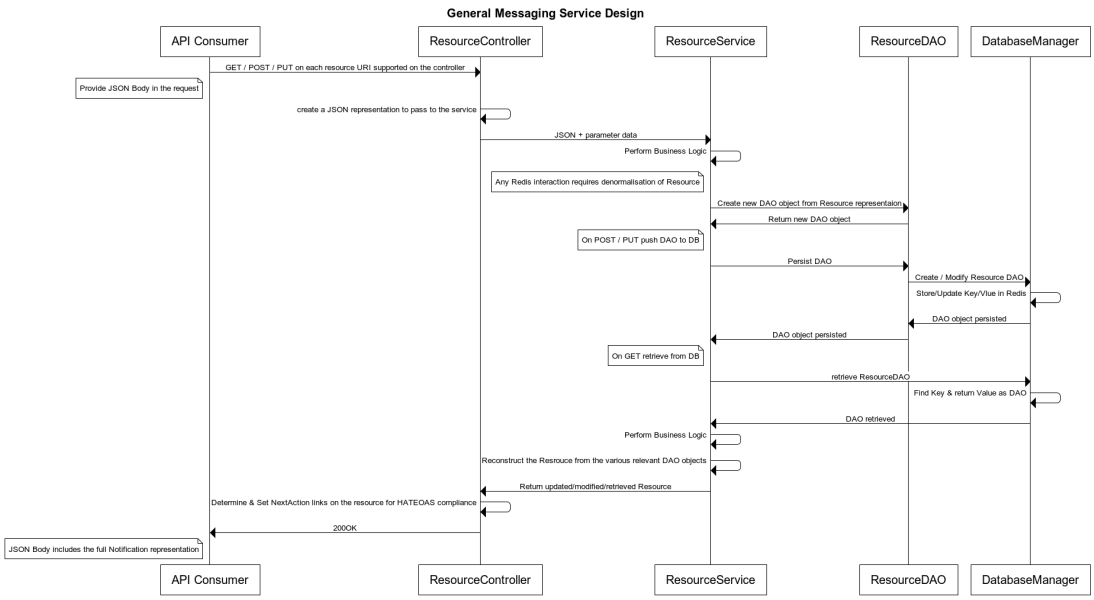


Figure 4.4 General pattern applied to all HTTP operations against a given Resource URI in the context of the four key design layers: Controller, Service, DAO & Database.

Other key technology & tooling choices for the prototype implementation are listed below & detail of how to get started is provided in the README.

- Java for delivery of cloud hosted messaging services via IntelliJ IDEA
- A local running Tomcat instance for webserver hosting
- A local running instance of redis-server for data storage & use of Jedis as the Java library to interact with it. Redis is an open source, advanced key-value cache & store.
- JMeter for load test & client simulation including additional plug-ing's for gathering usage reports.
- Firefox's HTTP Requester plug-in for issuing sample requests to the service
- Web sequence diagrams for generation of call flows

5. RESEARCH EVALUATION

The research evaluation that follows comprises two key elements:

1. An evaluation of how closely the Messaging API implemented satisfies REST constraints by assessing it against the four maturity levels described in the Richardson Maturity model (Richardson, 2009).
2. The second evaluation is the means by which the research hypothesis is validated & is based on data gathered during the execution of a suite of JMeter test plans designed to simulate the real-world traffic generated by Acme touch points leveraging the API implemented to deliver required functionality.

5.1 Evaluation of API 'RESTfulness'

There are 4 levels defined on the scale shown in figure 5.1 below. Reaching level 3 is necessary to be considered RESTful in this model. Each level is described below along with a review of how the services conform to each.

5.2.1 Maturity Model Level 0 (Not a RESTful framework)

Level 0 uses its implementing protocol (normally HTTP) like a transport protocol. That is, it tunnels requests and responses through its protocol without using the protocol to indicate application state. It will use only one entry point (URI) and one kind of method. In HTTP, this is normally the POST method.

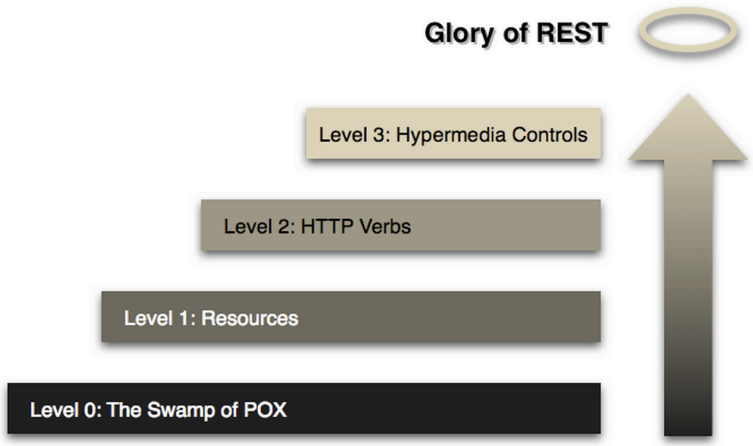


Figure 5.1 Richardson Maturity Model (Fowler, 2010)

- The Acme Messaging Services are built atop of the HTTP protocol as illustrated by the simple browser based request / response shown in figure 5.2 below
- All application state is represented as media in the JSON body. Though the phase one implementation does not support other media types, nothing precludes it from doing so.
- The services are designed to be fully REST compliant & therefore support more than a single URI entry point as outlined in the following sections.

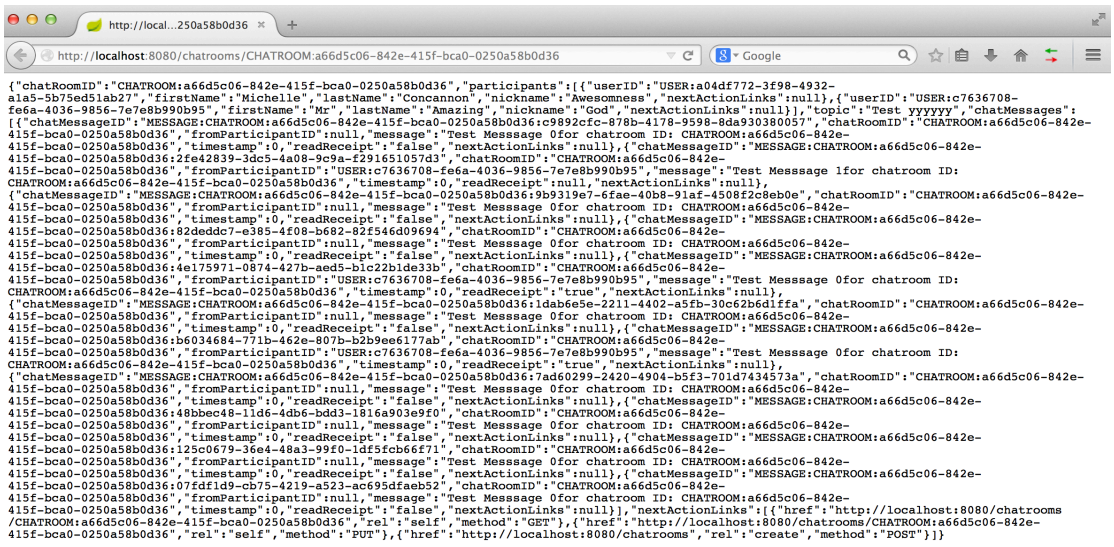


Figure 5.2 HTTP request / response

5.2.2 Maturity Model Level 1 (Resources)

When an API distinguishes between different resources, it achieves level 1. This level uses multiple URIs, where every URI is the entry point to a specific resource. Still, this level uses only one single method like POST.

- The Acme Messaging Services are designed around uniquely identifiable URI's There are 3 entry points to 3 distinct resources used to deliver the service: User, ChatRoom & Notification. There is a 4th resource defined, ChatMessage, but as it is only accessible from the ChatRoom resource does not warrant an entry point of it's own.
    - /users/{userAId} is distinct from /users/{userBId}
    - /chatrooms/{chatroomAId} is distinct from /chatrooms/{chatroomBId}
    - /notifications does not meet this criteria as Notification resources are used only to denote that some other resource has been modified. There is no

necessity to GET them a second time or modify them. This is an example of where pure REST constraints have been violated in an attempt to maintain state synchronization in a real time application based on synchronous HTTP alone.

- Each of these URIs support the POST operation & because the services are more evolved that the specifics of Level 1, they also support a series of other HTTP verbs which are a key premise of Level 2.

5.2.3 Maturity Model Level 2 (HTTP Verbs)

This level indicates that the API should use the protocol properties in order to deal with scalability and failures. For example, do not use a single POST for all, but make use of GET when you are requesting resources, and use the DELETE method when you want to delete a resource. Also, use the response codes of the application protocol used. Do not return a 200 (OK) code when something went wrong for instance.

- The partial implementation of Acme messaging services supports the three most commonly used HTTP Verbs: GET, POST & PUT.
- In all cases, GET is designed to be a safe operation - it can be invoked any number of times & return the same results each time. An important consequence is that it allows any participant in the routing of requests to use caching, a key element in making the cloud delivered service perform optimally.
- To create a ChatRoom, User or ChatMessage resource, the POST operation is supported – as defined in HTTP, the expected behavior is for POST to change some state. In the Acme implementation a POST supports creation (not modification) of only one resource at a time. There is no concept of a 'List' of resources in the original REST specification so batch operations are not generally considered 'RESTful'.
- To modify a ChatRoom, ChatMessage or User resource, the PUT verb is supported. The API makes a clear distinction between POST & PUT.
- The REST endpoints do not currently support DELETE / PATCH operations but these are not in fact widely used & therefore may introduce incompatibilities with clients that do not support them.
- The HTTP response codes are leveraged automatically to indicate various success / failure criteria. The implementation does not manually set response codes but

they are returned as an inherent part of the HTTP protocol as illustrated in figure 5.3.

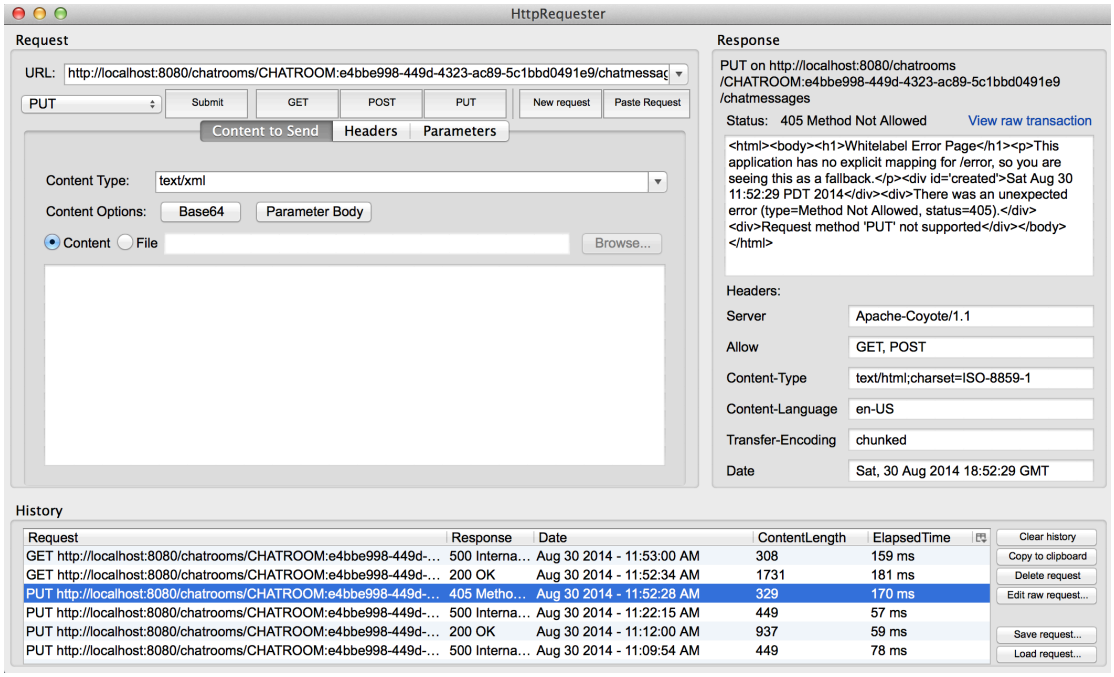


Figure 5.3 Leveraging HTTP Response Codes

5.2.2 Maturity Model Level 3 (Hypermedia Controls)

Level 3 introduces discoverability in HATEOAS, providing a way to make the API 'self-documenting'. True HATEOAS support means the client is able to navigate between states using self-descriptive message and no out-of-band information.

- The partial implementation of Acme messaging services is HATEOAS compliant in so far as every endpoint that successfully returns a resource representation also includes a set of ActionLinks within that resource to allow the consumer determine the next viable action. See the example provided in Figure 5.4 below.
- ActionLinks comprise three sub elements listed below in an attempt to provide sufficient metadata or hints for the client to infer what is feasible as the next action.
    - a href containing the <link> element,
    - a well known relationship provided in a <rel> e.g. self,
    - a supported method.

This brief analysis proves that the Messaging API is designed to a standard deemed 'RESTful' according to the Richardson Maturity Model. But the collaboration industry & development community at large has a more important question. How performant is it & what are the implications on cloud hosted resource usage?

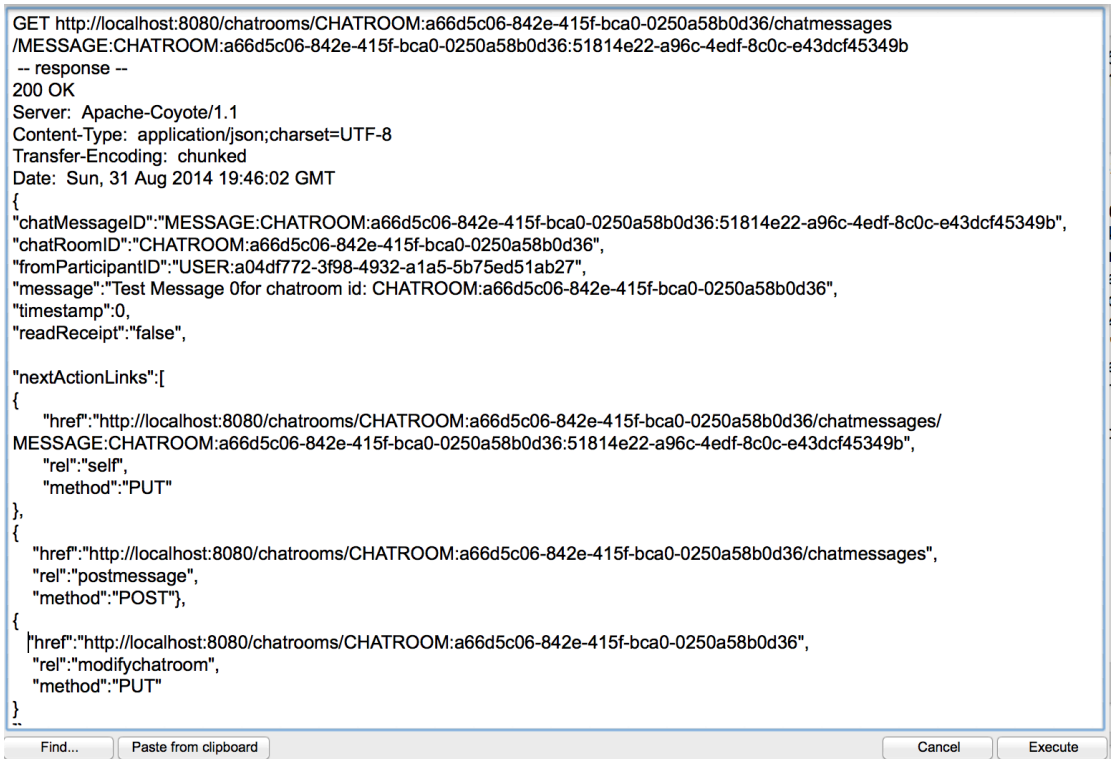


Figure 5.4 Example of Messaging Service's compliance to HATEOAS incl. the ActionLinks specified

5.2 Application Performance Evaluation

In order to validate the research hypothesis on a scale that could be reasonably extrapolated, two distinct JMeter test plans were developed in order to compare & contrast the results.

- Both simulations mode the anticipated traffic generated by their respective call flows (described below) for 10 distinct Acme users by running 10 threads in parallel over a 30 minute sample window.
- Both test plans make best effort to approximate a consuming clients implementation by making clear assumptions where a server implementation is not available in order to simulate the following subset of requirements:
    - User initiated usage e.g. creating a chat room with a new user (US13)
    - User response to usage by other Acme Users e.g. replying to a new chat message (US14) or marking that message 'read'
    - Some mechanism that preserves state synchronization & notifies the client of new events the end user needs to be aware of e.g. knowing when a new message has arrived & when a message they sent has been read by the remote user (US15, US18)

- The metrics gathered for each independent test suite for the purposes of this analysis include server hits per second & bytes throughput over time. A summary of traffic per endpoint is also provided for both overall Test Plans.
- Note that during these tests, the Acme Messaging services ran on 'locahost' & as such, network latency is not factored into the comparative analysis performed.

5.2.1 REST Compliant Test Plan

The first test plan simulates the system load in an implementation that leverages the RESTful API assessed in section 5.1 above. In this core version, three test suites run in parallel in order to demonstrate the key workloads of a solution based completely on REST over HTTP i.e. a solution which relies entirely on a request / response based, synchronous architecture. The various different types of processing that a single agent will perform to provide messaging services to its end users include:

1. A main thread simulating general usage initiated adhoc by a user
2. A background process which is constantly polling for new notifications
3. A thread which gets spawned to take action as new notifications are received

One of the JMeter Test Plans is illustrated in Figure 5.4 below & the call sequence diagrams for each of these 3 test suites, including the results generated follow. Both the test plans & a detailed list of reports generated are located in the git repository referenced in chapter 4 for closer analysis.

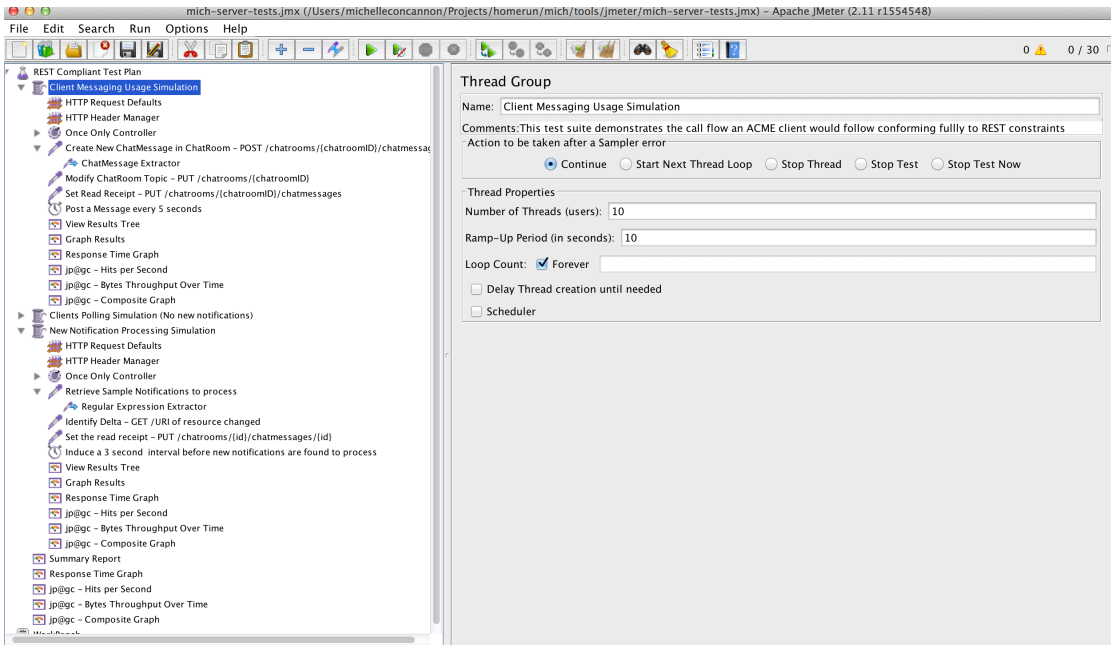


Figure 5.5 REST Compliant JMeter Test Plan incorporating three workloads

5.2.1.1 Adhoc Usage Simulation

The sequence of actions performed to emulate 'normal operation' in the JMeter test suite includes the three steps below being executed every 3 seconds.

- Posting a new Chat Message to a pre-existing ChatRoom

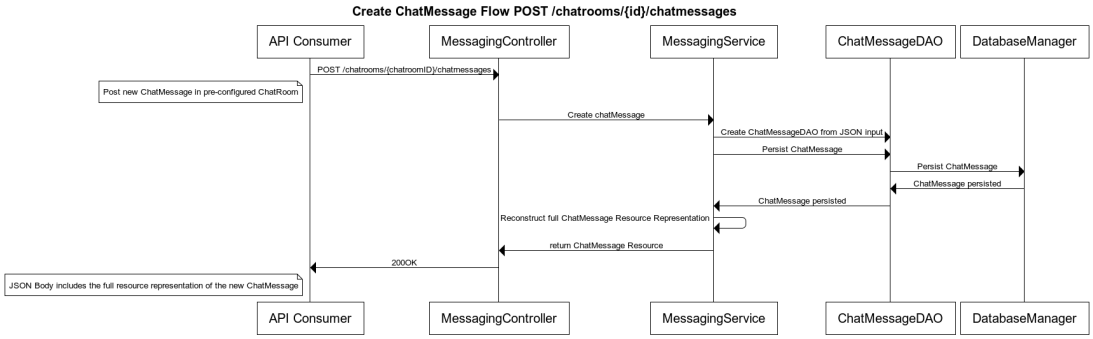


Figure 5.6 ChatMessage creation Call Flow – step 1 in adhoc usage simulation

- Modifying the ChatRoom topic

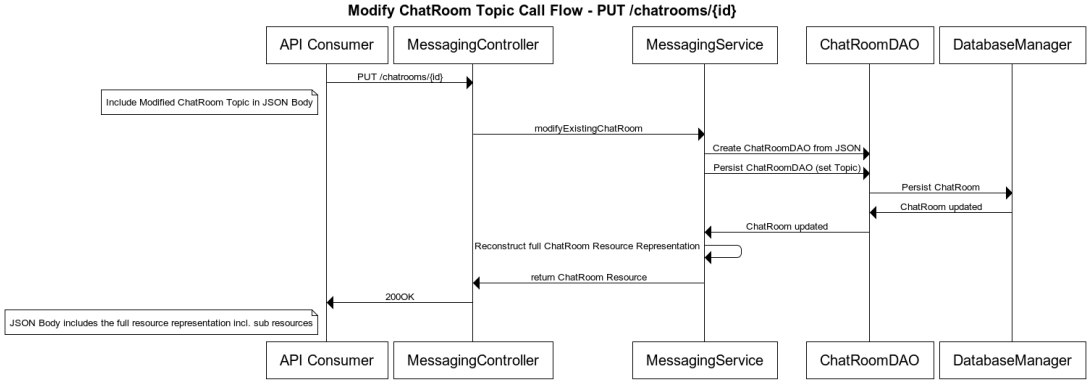
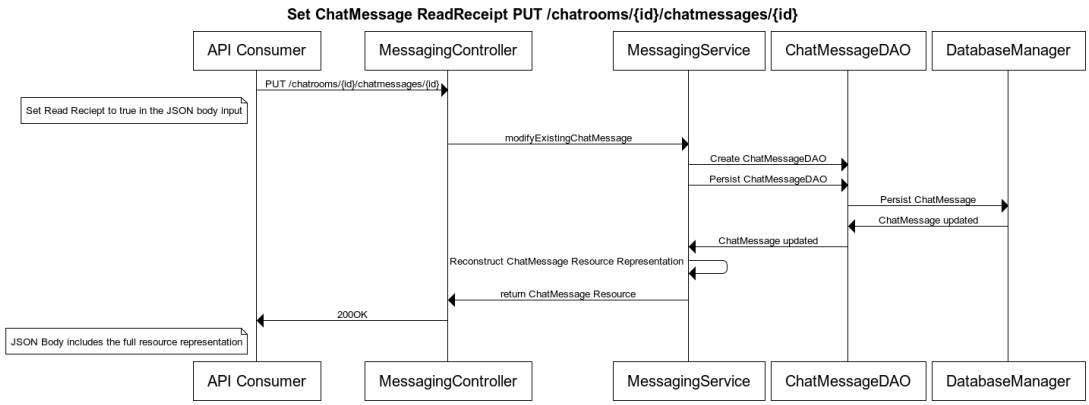


Figure 5.7 Modify ChatRoom Topic Call Flow – step 2 in adhoc usage simulation

- Setting the Read Receipt for the Chat Message created



- *Figure 5.8 Set ChatMessage Read Receipt Call Flow – step 3 in adhoc usage simulation*

The data shown below summarizes the server load in terms of request per second & throughput for the adhoc usage simulation described above.

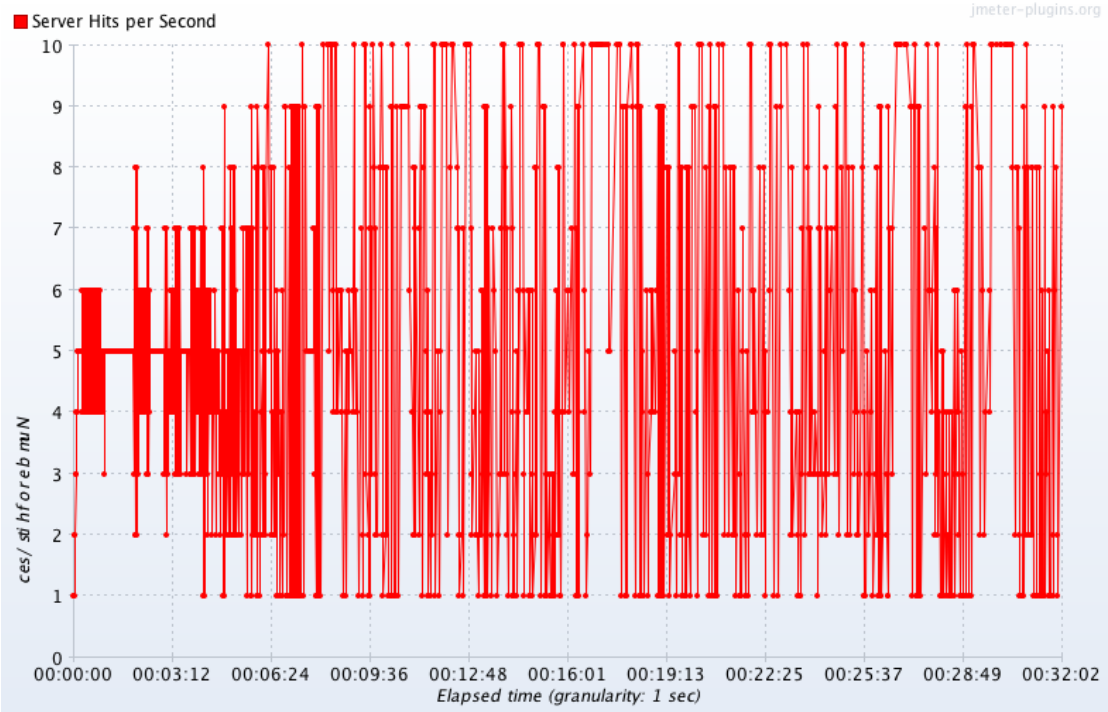


Figure 5.9 Server Hits per second modeling general usage of 10 users over 30 minutes

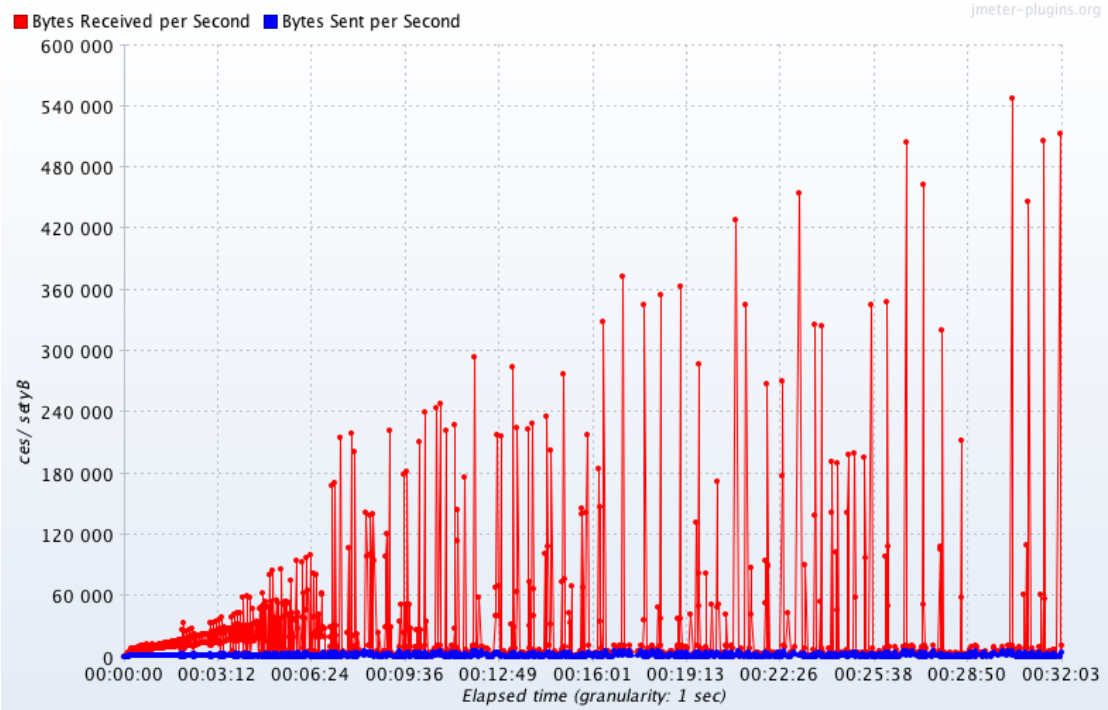


Figure 5.10 Bytes throughput simulating general usage of 10 users over 30 minutes. The more chat messages that get posted, the ChatRoom resource representation grows over time, hence the increase in bytes received.

5.2.1.2 Polling Simulation

The polling simulation is a constant loop of GET requests against the /notifications endpoint. No Notification resources are returned during this simulation, it's only purpose is to show the traffic that it can generate by simulating only ten users polling every 2 seconds (an industry standard polling interval for near real time). This is best observed in the summary chart at the end of this section where polling is proven to be the most resource intensive operation of this test plan.

The data shown below summarizes the server load & throughput for this simulation.

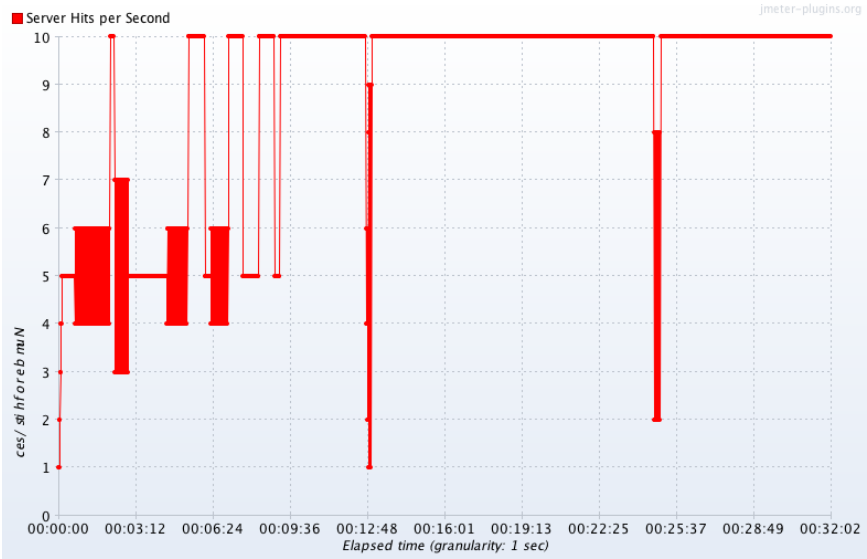


Figure 5.11 Server Hits per second modeling polling of 10 users with a 2 second polling interval over 30 minutes

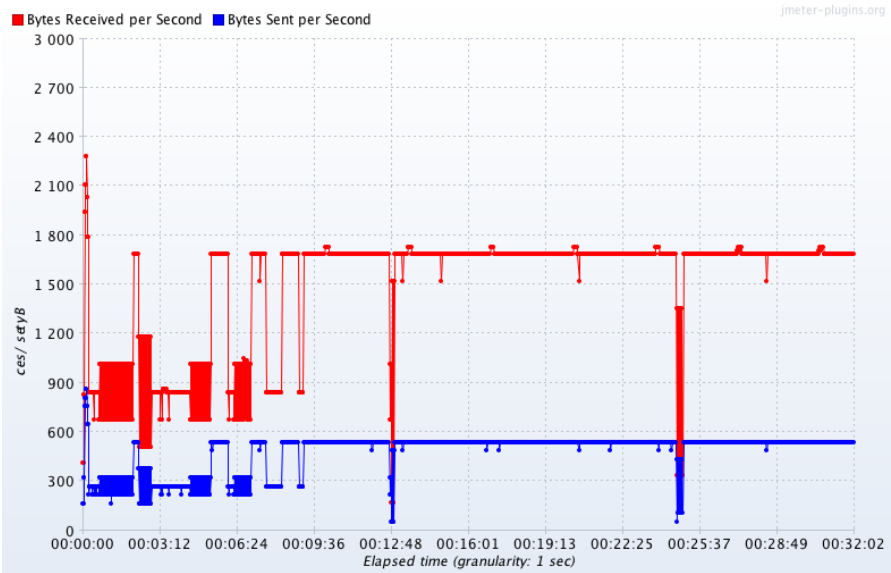


Figure 5.12 Bytes throughput simulating polling by 10 users at an interval of 2 seconds over 30 minutes

5.2.1.3 Notification Processing Simulation

The call flow for notification processing is similar to that shown above for adhoc usage except that in a real world example such as Acme, it is happening in parallel in the background while the user is proactively working on some action they initiated themselves in the foreground e.g. they are writing a new message while receiving one from a remote party, the response to which from the applications perspective is to place a 'new message' badge on the appropriate Chat Room. There are 3 basic steps in this load simulation & a 3 second interval between each execution of the sequence.

- Receipt of a new Notification resource, which has the detail of the type of resource modified & a unique URI link to retrieve it. In this example, every notification is a new ChatMessage being posted to a ChatRoom.

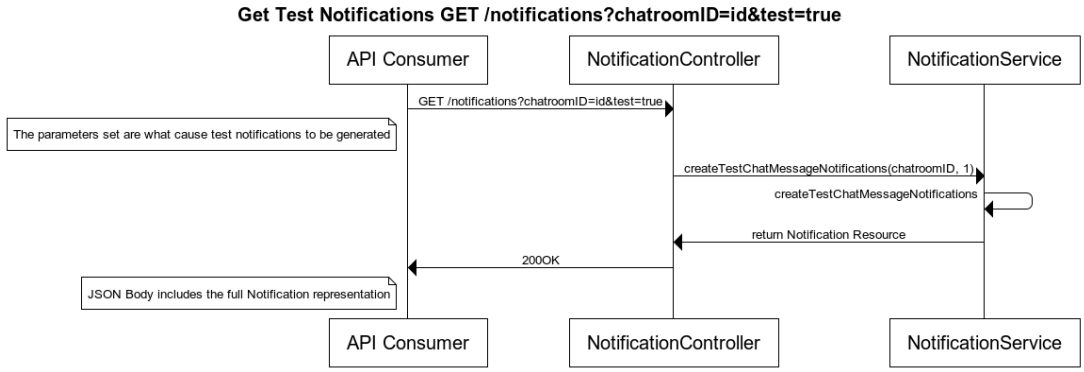


Figure 5.13 Notification creation Call Flow – step 1 in notification processing simulation.

- A GET operation on the URI received in the notification so the consuming client can determine what the delta is & perform the correct processing

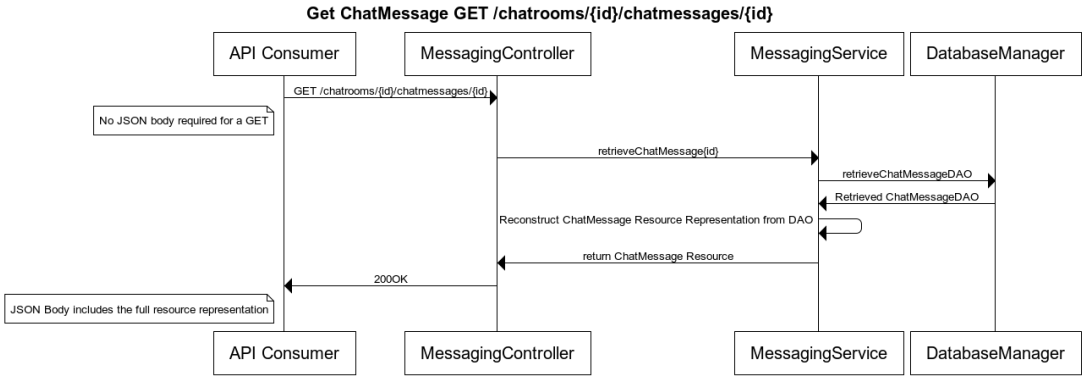


Figure 5.14 Get ChatMessage CallFlow to identify delta – step 2 in notification processing simulation.

- A simulation of one example of that processing which is setting the read receipt on the new ChatMessage notification received.

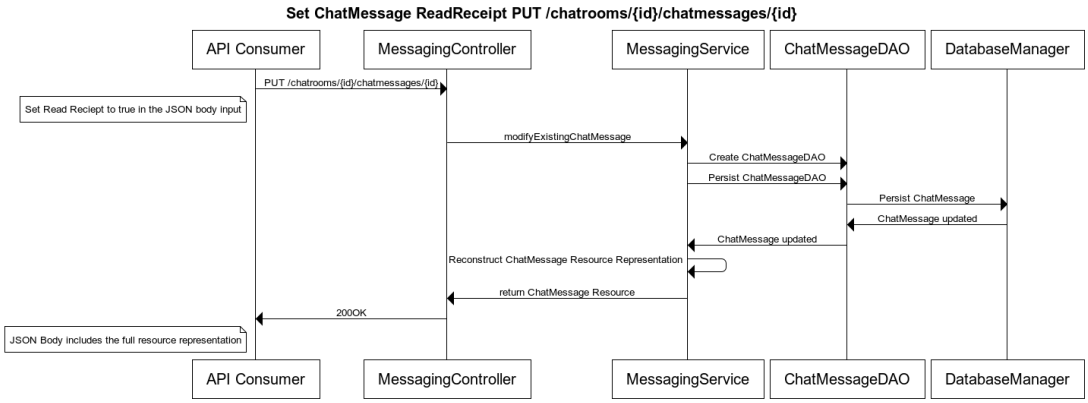


Figure 5.15 Set ChatMessage Read Receipt CallFlow– step 3 in notification processing simulation.

The result data shown below summarizes the server load & throughput for this part of the simulation.

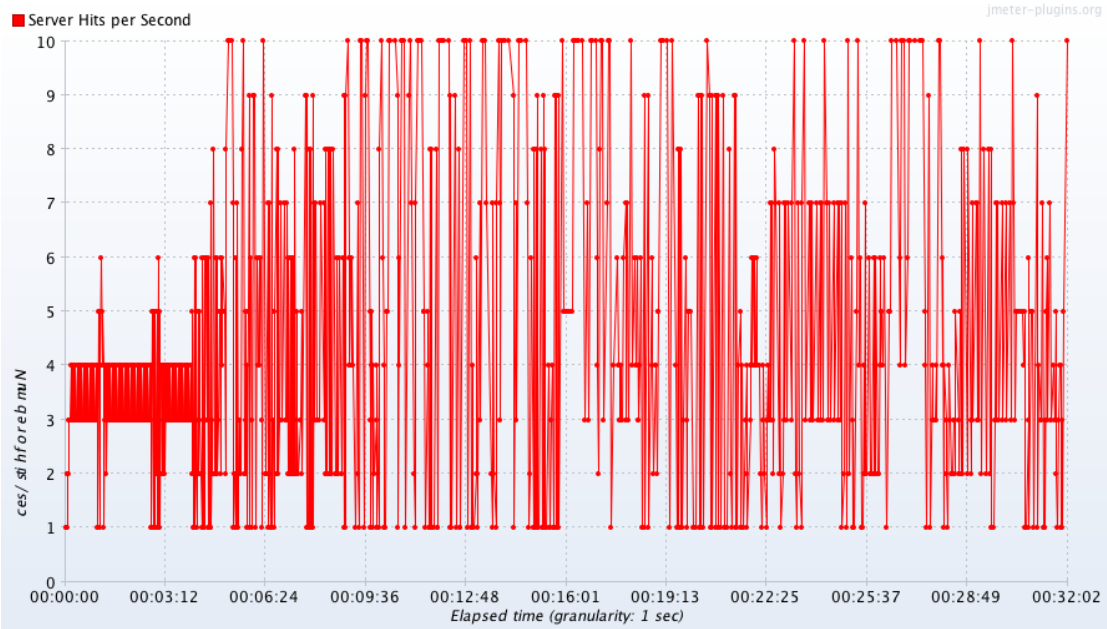


Figure 5.16 Server Hits per second modeling new notification processing for 10 users with a 3 second interval between new notifications over 30 minutes

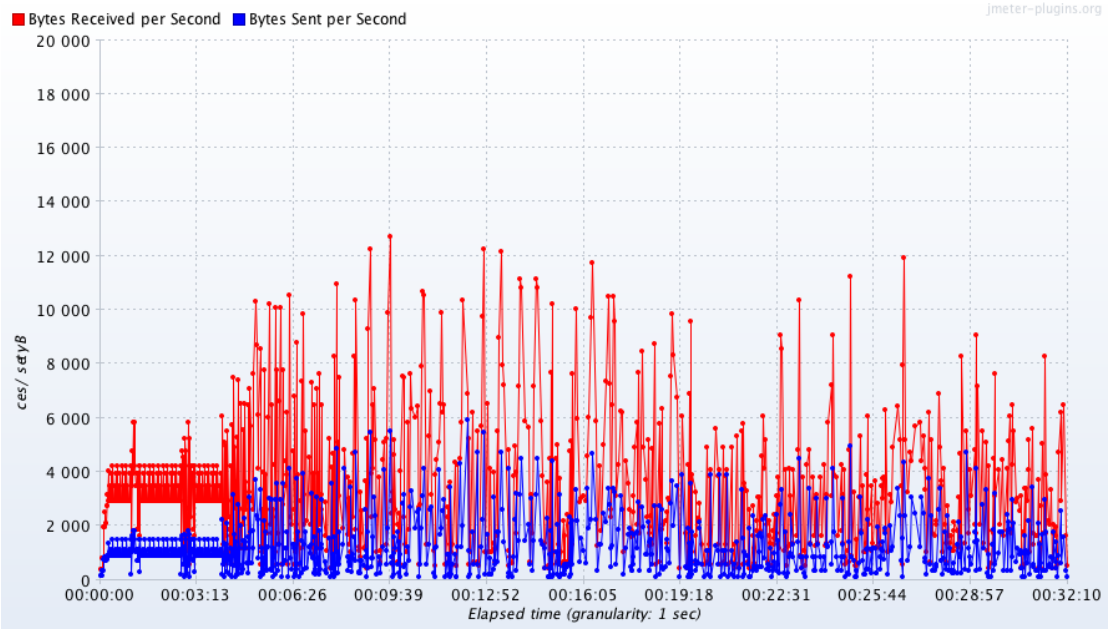


Figure 5.17 Bytes Throughput modeling new notification processing for 10 users with a 3 second interval between new notifications over 30 minutes

5.2.1.4 Overall 'REST Compliant' Simulation

The data shown below summarizes the total server load & throughput for this 'REST Compliant' simulation.

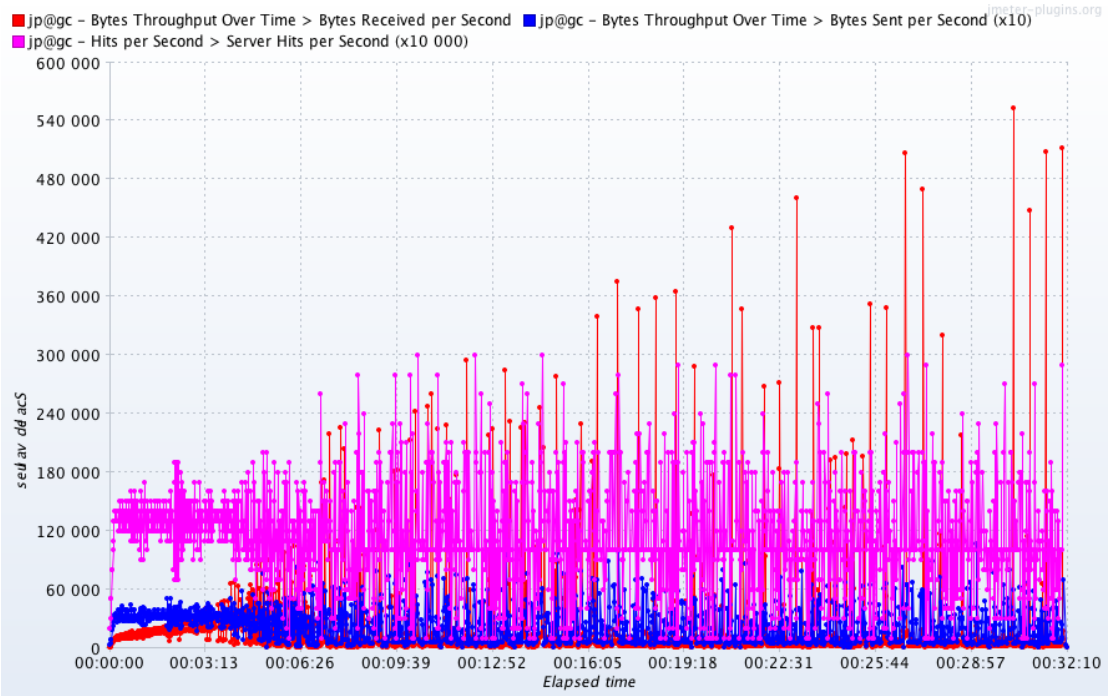


Figure 5.18 Composite View of Total Server Load & Throughput for Test Plan A

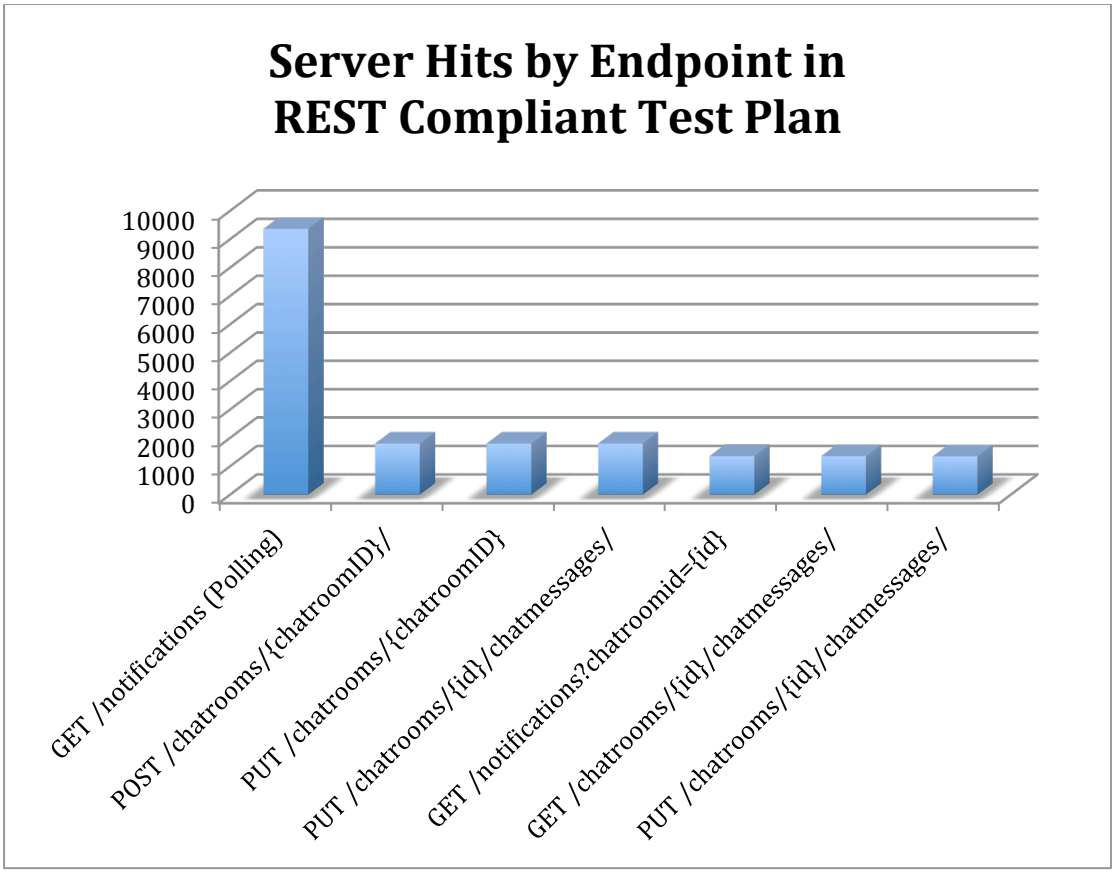


Figure 5.19 Breakdown of server load by endpoint for Test Plan A

The bar chart above generated from a subset of JMeter summary data from this run illustrates endpoints generate the most traffic in this test plan. As expected, the cost of polling in terms of overall server load is a significant overhead.

5.2.1 The 'Less RESTful' Approach

The second test plan presented provides some basic insight into the potential for reduction in network traffic & by implication, a reduction in the resources required in the cloud based infrastructure on which the service runs, by adopting a 'less RESTful', approach to the implementation. This is demonstrated by building one additional endpoint on the Messaging service that explicitly violates the generally accepted principle that resources should only be created or modified via the URI that identifies the resource in question. The approach implies that the API is now in partial violation of level 1 on the Richardson Maturity model (even though each resource still has it's own URI). Consider the following concrete example.

The existing API supports a POST action for the three key Messaging resources: ChatRoom, ChatMessage & User. The ChatRoom resource representation contains a

list of ChatMessage & User resources, URI's for which both support the 'POST' operation independently.

The POST /chatrooms endpoint only creates the parent elements of the ChatRoom resource & will ignore any ChatMessage or User sub resources contained in the JSON body. It is enforced by the business logic that the API consumer will create / modify those sub resources by making independent POST requests to their associated URI's. Similarly, the existing API leverages 'PUT' as the means to modify only a single resource accessible via it's specific URI. These constraints can be observed in the call flows shown for 5.2.1 above.

Test Plan B relaxes these constrains & leverages one newly endpoint, overloaded to create OR modify a ChatRoom *as well as* one of it's sub resources, a ChatMessage, via a single POST operation. This is the example that will realize the maximum gains in server round trip time for the specific examples measured & is being used in this research as a means to highlight the impact of making such modifications on the cloud hosted resources on which these services run. However, there are many levels on which this type of semantic aggregation could be applied including (but not limited to):

- Maintaining the semantic separation of POST & PUT so POST only *creates* new ChatRoom resources as well as it's sub resources. PUT continues to modify them but can also perform modifications across the sub resources associated with a given ChatRoom.
- Support batch operations on the existing API so that POSTing to a ChatRoom can support the creation of multiple ChatRooms in one go, similarly on the other resources.

Ultimately these decisions are best made in the context of the application domain as well as the degree to which the team at Acme believe they can maintain an intuitiveness of API design for 3rd party developers. The tradeoff is increased documentation, some baseline of which this research has deemed necessary in REST based API's regardless of claims that HATEOAS eliminates this requirement. Overloading these types of operations can also cause complexity as is visible in the

source code but if there are meaningful performance gains, this is a trade off worth considering.

As observed in figure 5.19 above, the major traffic generated is test plan A is from the /notifications endpoint on which all clients poll every two seconds. Test plan B also removes this test suite in order to demonstrate that introducing a different mechanism for state synchronization - some non HTTP based communication protocol (e.g. sockest), which completely eliminates the requirement to poll, is one of the quickest wins by way of overall system performance.

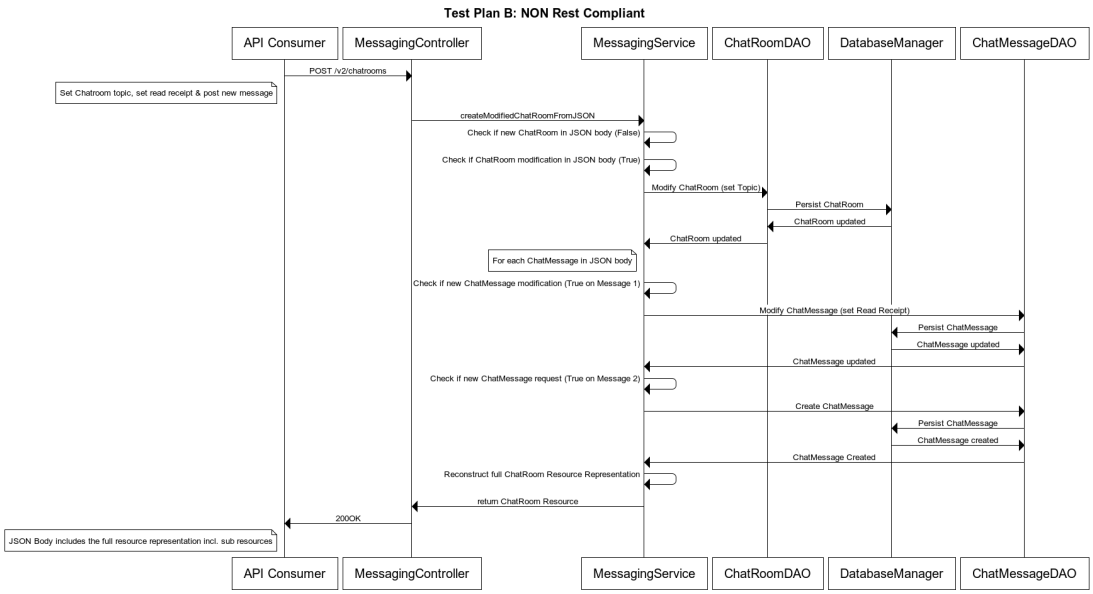


Figure 5.20 Modified Usage Simulation Call Flow for Test Plan B where 3 separate HTTP requests are collapsed down to 1 with increased complexity in the backend

The modified call flow shown above & resulting data for the same steps performed in the initial adhoc usage simulation described in 5.2.1.1 above, illustrates the significant reduction in traffic that can be achieved by these modifications. The Notification Processing simulation is unchanged from Test Plan A as irrespective of how a consuming client is made aware of new notifications, they will need to be processed. The Polling simulation is removed & assumed to be replaced by a more optimal design as noted above.

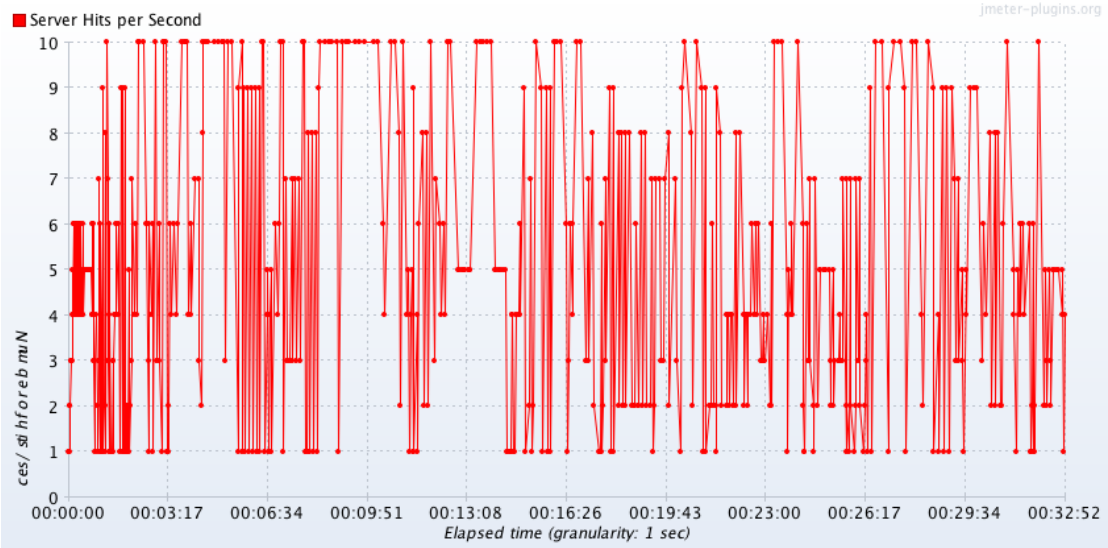


Figure 5.21 Server Hits per second for simulated usage processing in test plan B where the same workload is executed over the same 30 minute time period. In this version it exhibits significantly less concentration of his over the test period than seen in figure 5.9 above

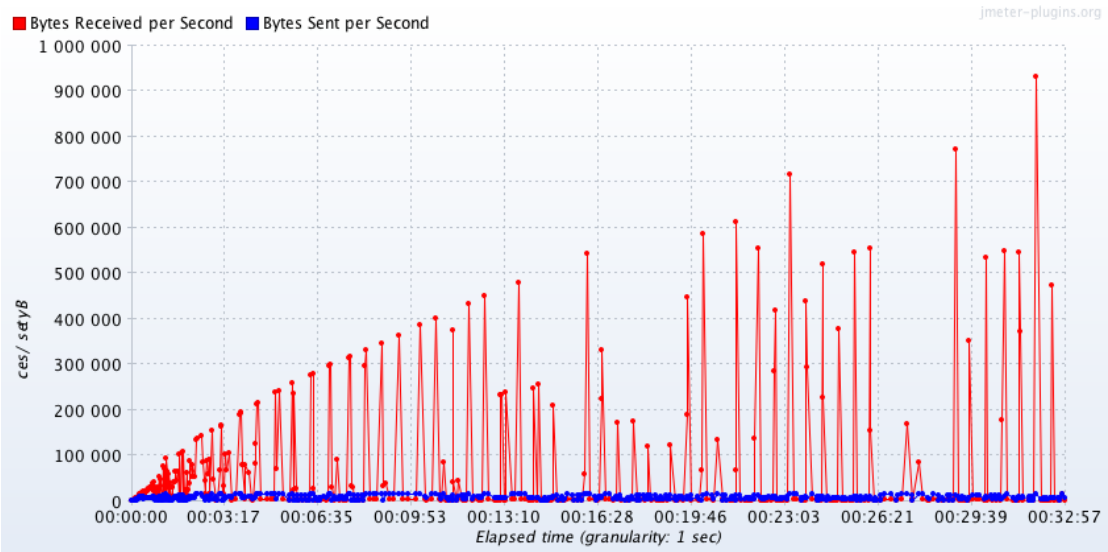


Figure 5.22 Bytes throughput in the test plan B where the same usage workload is executed over the same 30 minute time period.

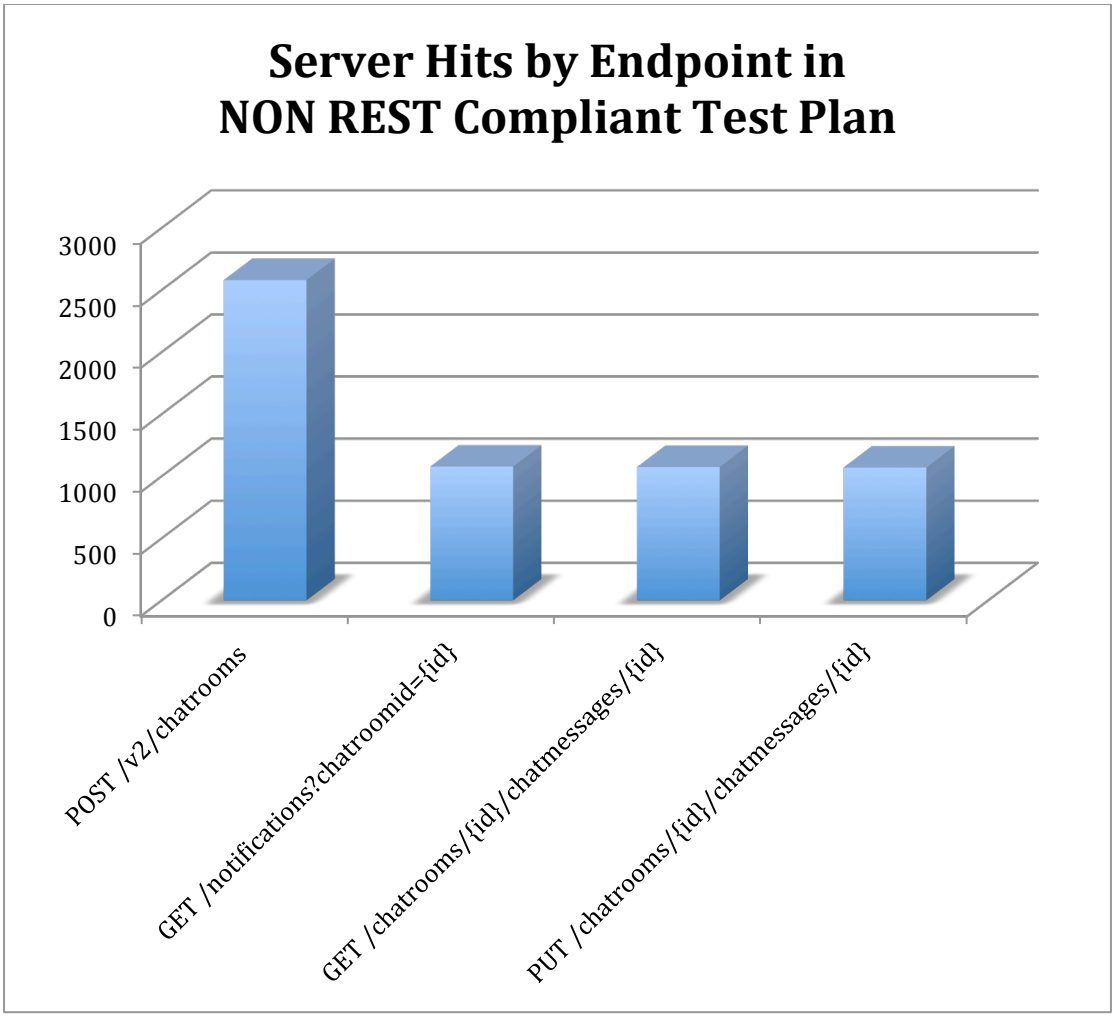


Figure 5.23 Bytes throughput in the test plan B where the same usage workload is executed over the same 30 minute time period. Total no. server hits is 5,868 v 18,882. ***Traffic is reduced by 1/3.***

Chapter Six

6. CONCLUSION & RECOMMENDATION

Cloud delivered services & deployment models have unquestionable benefits. Where leveraged in a fashion similar to the Acme use case above to balance immediate & long-term goals, they can dramatically increase time to market & lower the cost of infrastructure & development

REST has been proven to offer the fundamental building blocks for architecting a cloud hosted, highly distributed service. Adherence to the REST constrains defined by Fielding (2000) will guarantee support for the subset of key characteristics of a cloud delivered service, thus enabling operation in a highly distributed environment. Following the constraints also facilitates near real time delivery of a developer platform already familiar to the software development community, which can therefore be quickly & effectively leveraged by internal & external 3rd parties to unlock significant business value by LOB integrations.

However, the research presented also demonstrates some key shortcomings of REST in the context of enterprise scale, real-time collaboration solutions & this can be easily extrapolated to other industries.

1. Even strict adherence to HATEOUS does not eliminate the need for documentation describing the semantic meaning of a service & by implication, produces an unavoidable element of coupling between client & server. To use a service effectively you need to know more than it's URI structure including:

   - What query parameters it can accept & the possible values
   - The structure of the JSON/XML/whatever documents you need to send in your POST/PUT/etc requests
   - The structure of the response sent by the server
   - The possible errors that might occur

Based on the above, HATEOAS only solves a tiny fraction of the discoverability and coupling problems & to avoid breaking clients, you still need to version your API.

2. Strict adherence to the broader set of REST constraints to preserve the purity of separation between resource, representation & the rules by which the former get manipulated, **introduces a performance degradation which lacks necessary pragmatism when performing time sensitive operations**. Two such demonstrated examples include:

   - The requirement to perform any valid HTTP operation at the appropriate level in the resource hierarchy in order to perform direct manipulation of the resource. This induces the property of multiple server round trips & a more pragmatic alternative may be to manipulate a series of resources from the operation at the parent. The analysis performed in chapter 5 proves that by violating this constraint it is possible to reduce server load by over 50% (from 5385 server hits in test plan A to 2579 in test plan B for the same workload).

   - The performance issues inherent in a pull model that requires ongoing polling by a client is also unrealistic for time sensitive media operations such as actively 'muting' a call, identifying when other participants in a meeting are muted or detecting client network disconnects during an active meeting. Domains other than call control as proven here, would also realize performance gains by introducing some push mechanism side by side with the request response architecture which REST is based on when a server detects that a resource has been modified by some other user agent, thus rendering other interested parties copies stale.

3. Complete Reliance on REST alone for real-time solutions for the purposes of delivering an API that is self-describing also overlooks the performance gains that could be realized by complementing a REST based architecture with a socket connection for delivery of notifications. This is an area for future research as we see large corporations debate over short polling, long polling & socket based solutions.

4. State Synchronization is a major & challenging problem where each request is stateless & can independently manipulate resources in cases where the client has to rely on poling to refresh stale data. The window of opportunity for race

conditions to exist is bigger & another reason why a more time sensitive approach to state synchronization is warranted.

Key recommendations from this research to any company seeking to migrate to, or build a set of cloud delivered services using REST that are similar in nature to the collaboration specific examples presented in this thesis are summarized below. They are motivated by the conclusions derived above from the work presented in Chapter 4 & 5 & first hand observation of these issues in a leading industry collaboration organization.

1. Build cloud hosted services & leverage a PaaS or SaaS based solution where feasible
2. Leverage REST for cloud services to support the key premise that components in a distributed cloud are loosely coupled with the consuming client thus exhibiting a high degree of fault tolerance.
3. Avoid the assumption that documentation is not required in a REST based API & plan to incorporate it, even where the API is believed to be fully REST compliant.
4. Version the API in a reasonable cadence to avoid significant challenges with supporting backward compatibility
5. Implement a pub / sub (socket) based protocol side by side with a HTTP request response based REST implementation to realize significant performance gains.

REFERENCES

Amanatullah, Y, Lim, C, Ipung H.P, Juliandri, A. (2013) "Toward Cloud Computing Reference Architecture: Cloud Service Management Perspective", Proceedings of the 2013 International Conference on ICT for Smart Society (ICISS), Jakarta, 13-14 June 2013, pp. 1-4.

Apigee (2014) The Digital Business Platform. Available from: http://apigee.com/about/ [15 August 2014]

Avanade (2013) Global Survey: Is enterprise social collaboration living up to it's promise? May 2013. Available from: http://www.avanade.com/Documents/Resources/social-collaboration-global-study.pdf [01 August 2014]

Birman, K.P. (2005) Reliable Distributed Systems: Technologies, Web Services and Applications, 1st ed., New York: Springer Science + Business Media, Inc.

Booth, D, Haas, H, McCabe F, et al. (2004) W3C Working Group Note 11: Web Services Architecture. Available from: http://www.w3.org/TR/ws-arch/wsa.pdf [13 July 2014]

BT & Cisco (2013) BT & Cisco 'Beyond your Device' Research. Available from:http://business.bt.com/assets/pdf/it-support-and-security/BT_Cisco_Beyond_Your_Device_research.pdf [4 August 2014]

Business Hangouts (2014) Bringing Google Hangouts to the Enterprise (Not a Google Product). Available from: https://business-hangouts.com/. [4 August 2014]

Clarke, J, Stikeleather, J, Fingar, P. (1996) Distributed Object Computing For Business in Next Generation Computing: Distributed Objects for Business, eds. P. Fingar, D. Reid, & J. Stikeleather 1st ed., SIGS Books & Multimedia, New York, New York.

Cloud Foundry (2014) Cloud Foundry Foundation. Available from: http://cloudfoundry.org/index.html [15 August 2014]

Cloud Security Alliance (CSA) (2011) Security guidelines for Critical Areas of Focus in Cloud Computing v3.0. Retrieved from https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf

Coetzee, L, Eksteen, J. (2011) "The Internet of Things – a promise for the future? An Introduction", Proceedings of the IST Africa Conference, Gaborone, 11-13 May 2001, pp.1-9.

Escalera, M.F.P, Chavez, M.A.L. (2012) "UML Model of a Standard API for Cloud Computing Application Development", Proceedings of the 2012 9th International Conference on Electrical Engineering, Computing Science & Automatic Control (CCE), Mexico City, 26-28 May 2012, pp.1-8.

Fielding, R.T. (2000) Architectural Styles and the Design of Network-based Software Architectures, PhD thesis, University of California, Irving.

Fielding, R.T. (2008) Untangled: musings of Roy T. Fielding.28 October 2008. *Roy T. Fielding: REST APIs must be Hypertext driven*. Available from: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven. [6 July 2014]

Fowler, M. (2010) Introduction to martinfowler.com. 18 March 2010. *Richardson Maturity Model: steps toward the glory of REST*. Available from: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven. [6 August 2014]

Gambi, A, Pautasso, C. (2013) "RESTful Business Process Management in the Cloud", Proceedings of the 2013 Principles of Engineering Service-Oriented Systems (PESOS), 2013 ICSE Workshop on, San Francisco, 26-26 May 2013, pp. 1-10.

Gartner (2011), Gartner Executive Programs Worldwide Survey of more than 2,000 CIOs Identifies Cloud Computing as Top Technology Priority for CIO's in 2011, 21 January 2011. Gartner Newsroom. Available from: http://www.gartner.com/newsroom/id/1526414 [01 August, 2014]

Gartner (2013), Top 10 CIO Business & Technology Priorities in 2013, Available from: http://www.gartnerinfo.com/sym23/evtm_219_CIOtop10%5B3%5D.pdf [01 August, 2014]

Gens, F. (2013) IDC Predictions 2013: Competing on the 3rd Platform. Available from: http://www.idc.com/research/Predictions13/index.jsp;jsessionid=D2F6BF7DE6869A9FAF02DA1C659C633F#.UVk_1Fd6o5w. [15 Aug, 2014]

Hafner, K. & Lyon, M. (1996) Where Wizards Stay Up Late: The Origins of the Internet, 1999 ed., New York: Simon & Schuster.

Heineman, G. & Councill, W. (2001) Component-based software engineering: Putting the pieces together, 1st ed., Boston: Addison-Wesley Professional.

High, P. (2013) Forbes. 14 October 2013. Gartner: Top 10 Strategic Technology Trends For 2104. Available from: http://www.forbes.com/sites/peterhigh/2013/10/14/gartner-top-10-strategic-technology-trends-for-2014/. [01 August 2014]

Hunter II, T. (2013) Code Planet. 31 December 2013. *Thomas Hunter II: Principles of Good RESTful API design*. Available from: http://codeplanet.io/principles-good-restful-api-design/. [5 July 2014]

Hwang, K, Fox, G.C, & Dongarra, J.J (2012) Distributed & Cloud Computing: From parallel processing to the Internet of Things, 1st ed., Waltham: Elsevier

Humble, J. (2014) ThoughtWorks. 13 February 2014. *Jez Humble: The Case for Continuous Delivery.* Available from: http://www.thoughtworks.com/insights/blog/case-continuous-delivery [16 August, 2014]

Kavis, M. (2014) Architecting The Cloud, 1st ed., Hoboken: John Wiley & Sons, Inc.

Knupp, J. (2014) Everything I know about Python... 03 June 2014. *Jeff Knupp: Why I Hate Hateoas.* Available from: http://www.jeffknupp.com/blog/2014/06/03/why-i-hate-hateoas/ [14 July 2014]

Liu, M.L. (2004) Distributed Computing: Principles and Applications, 1st ed., Addison-Wesley

Lunden, I. (2014) Tech Crunch 2014, Gartner: Device Shipments Break 2.4B Unit In 2014, Tablets To Overtake PC Sales In 2015. Available from: http://techcrunch.com/2014/07/06/gartner-device-shipments-break-2-4b-units-in-2014-tablets-to-overtake-pc-sales-in-2015/ [4 August 2014].

Mahajan, S. & Shah, S. (2013) Distributed Computing, 2nd ed., New Delhi: Oxford University Press

Mell, P. & Grance, T. (2011) The NIST definition of Cloud Computing Recommendations of the National Institute of Standards & Technology. Retrieved from http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf [01 August, 2014]

Meyer, B. (2000) Object-oriented software construction, 2nd ed., New York, NY: Prentice Hall

Millard, N.J. & Gillies, S. (2011) Workshift: The future of the Office. Available from: http://business.bt.com/assets/pdf/it-support-and-security/byod-whitepaper-the-future-of-the-office.pdf [4 August 2014]

Mollah, M.B, Islam, K.R, Islam, S.S. (2012) "Next generation of computing through cloud computing technology", Proceedings of the 2012 25th IEEE Conference on Computer & Electrical Engineering (CCECE), 2012, Montreal, 29 April – May 2 2012, pp. 1-6.

Qiu, R. (2013) Business-Oriented Enterprise Integration for Organizational Agility, 1st ed., Hershey: IGI Global

Richardson, L. (2009) Crummy, the webspace of Leonard Richardson. 16 January 2009. *Act Three: The Maturity Heuristic.* Available from http://www.crummy.com/writing/speaking/2008-QCon/act3.html [14 August 2014]

Robinson, I. (2006) Consumer-Driven Contracts: A Service Evolution Pattern. Available from: http://martinfowler.com/articles/consumerDrivenContracts.html [5 July 2014].

Rosenberg, J. (2014) Cloud & Virtualization: More than just a silver lining. VP. CTO Cloud Collaboration Group, Cisco Systems. [Unpublished]

Salesforce (2014) Connect with everything that's important to you. Available from: http://www.desk.com/features/salesforce [15 August 2014]

Saylor, M. (2012) The Mobile Wave, 1st ed., New York: Vanguard Press

Shapiro, O. (2014) The Vidyo Blog. 06 February 2014. *Ofer Shapiro: VidyoH2O for Google + Hangouts is a WebRTC Product*. Available from http://blog.vidyo.com/vidyo-news/vidyoh2ogooglehangouts/ [04 August 2014]

Tuli A, Hasteer N, Sharma M. & Bansal A. (2013) "Exploring challenges in Mobile cloud computing: An overview", Proceedings of the 4th International conference on The Next Generation Technology Summit, Nodia, India, 26-27 Sept 2013, pp. 496-501.

Weinman, J. (2012) Cloudonomics: The Business Value of Cloud Computing, 1st ed., Hoboken: John Wiley & Sons, Inc.

Weinman, J. (2008a) BusinessWeek. The 10 laws of Cloudnomics', Sept 6 2008. Available from: www.businessweek.com/technology/content/sep2008/tc2008095_942690.htm [01 August 2014].

Weinman, J. (2008b) GigaOM.com. The 10 laws of Cloudnomics', Sept 7 2008. Available from http://gigaom.com/2008/09/07/the-10-laws-of-cloudonomics/ [01 August 2014].

APPENDIX A

User Story ID	User Story Name
US9	**ACME End User Requirements**
US13	Messaging: Create a 1:1 Persistent Chat Room
US16	Messaging: Modifyable Topic for a 1:1 Persistent Chat Room
US14	Messaging: Post a message in an existing Chat Room
US15	Messaging: Get notified of a new message in an existing Chat Room
US18	Messaging: Read Receipt Notification in a 1:1 Persistent Chat Room
US17	Messaging: Ability to access all messages in the Chat Room at any time
US19	Live Meeting: Invite a Participant in the Chat Room to Join a 1:1 Real-Time Meeting
US20	Live Meeting: Get Notified of an invitation to a 1:1 real time meeting
US21	Live Meeting: Accept an invite to a 1:1 real time meeting
US24	Live Meeting: Ability to cancel an invite to a 1:1 real time meeting
US23	Live Meeting: Ability to leave a 1:1 real time meeting
US22	Live Meeting: Decline an invite to a 1:1 real time meeting
US7	**ACME Dev Ops Requirements**
US32	Development Platform: Cloud Hosted development platform
US29	Usage Analytics: Understand Usage of Messaging & Meeting Services
US30	Reliability Metrics: Understand Reliability of Messaging & Meeting Services
US37	Effective Troubleshooting: Service log storage
US31	Alerts: Automated Notification of a Service Outage
US10	**ACME Performance Requirements**
US25	Live Meeting Set Up Time: <1 second live meeting set up
US27	Notification Delivery: <1 second delivery of new event notifications to remote chat room participant
US26	Message Delivery: Instant Message Posting to Chat Room on Sender
US36	Automate Performance Testing: Understand when some commit has degraded performance
US8	**ACME Deployment Requirements**
US34	Automated Testing: Guarantee production quality builds each commit

US33	Deploy Velocity: Support multiple service deploys per day
US11	**ACME Scalability Requirements**
US28	Dynamic Traffic Loads
US35	Service Throughput: Learn the Maximum Service Throughput
US12	**ACME 3rd Party API Requirements**
US38	Client Developer API: Consistent & Performant REST API
US39	3rd Party Developer API: Documented, Consistent REST API

NOTE: This table is exported from the Rally agile tool & accessible from https://rally1.rallydev.com/#/20912212734d/userstories?tpsV=qv%3A0

Username: mconcannusa@gmail.com, **Password**: gmitmsc123!