A Service Based Approach For Future Internet Architectures

Ram Kumar Ravikumar Santhi

A Service Based Approach for Future Internet Architectures

Doctoral Dissertation for the degree Philosophiae Doctor (PhD)
in Information and Communication Technology

UNIVERSITY OF AGDER
Faculty of Engineering and Science
2010

Doctoral Dissertations at the University of Agder 28

ISBN: 978-82-7117-685-3

ISSN: 1504-9272

Printed by the Printing Office, University of Agder

Kristiansand

Preface

I owe my deepest gratitude to my supervisor Prof. Frank Reichert for his encouragement, guidance and patience throughout the duration of my work on this thesis. His support enabled me to develop an understanding of the subject and I could not have imagined having a better advisor and mentor for my Ph.D study. My sincere thanks go to my co-supervisor Prof. Vladimir Oleschuk for his encouragement during my study and work at the university.

I thank my student colleagues at the University of Agder: Dmitry Umansky, Andreas Häber, Liping Mu, Yuanyuan Ma, Morten Bak, Cheelim Nge and Martin Choux for the stimulating discussions, for the much needed caffeine and for all the fun we have had in the last four years. I am indebted to many more of my friends, colleagues and students at UiA, whose company has made my duration of study memorable and fruitful. I also extend my gratitude to Prof. Andreas Prinz and Trine Tønnessen for their help in navigating the administrative and regulatory mazes.

Last but not the least, I would like to thank my parents, brothers and Rashmi for their understanding and patience with me during the ups and downs of my PhD life. Without their encouragement and support it would have been impossible for me to finish this work.

For my parents

They bore me, raised me, supported me, taught me and loved me.

and

Dedicated to

My little angel, Rayna.

Summary

Over the past three decades, the Internet has evolved from a point to point, open, academic network to an applications and services oriented critical infrastructure. The Internet has become a vital component of society today, from its humble origins as an academic research project, but still uses the one dimensional layered model as the underlying communication architecture. During this transition, numerous applications and usages of the network emerged that cannot be efficiently implemented by adhering to the original design tenets of the Internet. Some of these principles have been broken, others diluted and new ones are emerging to accommodate new paradigms.

Simultaneously, applications and services have been moving slowly but consistently towards a uniform model based on Service Oriented Approach (SOA). The shift towards abstract models, objects and services however is not supported by the underlying delivery platforms, especially the legacy Internet architecture. An architectural rethinking is necessary at the network level, to accommodate future services, applications and routing priorities. We argue that there is a pressing need to move towards a next generation network architecture built to natively support parallel processing of communication tasks, high level network resource abstraction, enhanced routing, privacy, Quality of Experience (QoE), heterogeneous networking etc. This new architecture should be manifested according to the principles of SOA to ensure interoperability, backwards compatibility and migration. We extend this approach by following SOA to make components more reusable and scalable. This thesis proposes a new communication architecture allowing integration of characteristics from several dimensions, in particular security, mobility and context-sensitivity. This new architecture integrates these dimensions in ways not feasible within the layered paradigm.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This chapter sets the stage for this thesis. The sections within this chapter describe the motivation for the research question and breaks down the background problems into subsections. A view of the the approach taken and the organization of the thesis is given in section 1.2 and a summary of the motivational issues are compacted into a table at the end of the chapter

The Internet, as a network of connected computers, came into existence in the 1970s [6]. There has not been any fundamental change in the architecture since then, even though the usage and possibilities have expanded beyond its initial scope. The early Internet interconnected a stationary set of nodes. The architecture and the protocols used were designed to accommodate the simple stationary nature. The designers wanted to build a network infrastructure to interconnect all the computers in the world together and provide a framework for yet unknown applications to be invented and run [7]. The nodes were well described by IP addresses that identified the nodes directly on the network. Routing protocols then took advantage of the static nature of the Internet.

The Internet has grown over the years with more powerful routers, faster backbones, faster processing at end points, QoS (although limited) and traffic shaping to accommodate newer models of traffic and faster access to customers. From a mere 15 hosts during the beginning of the 1970s [8], the number has presently passed the 1.8 billion mark in 2009 and steadily increasing Fig. 1.1. The exponential growth is more remarkable considering the growth of other aspects such as Internet Domains, Internet Networks etc. as can be seen in [1].

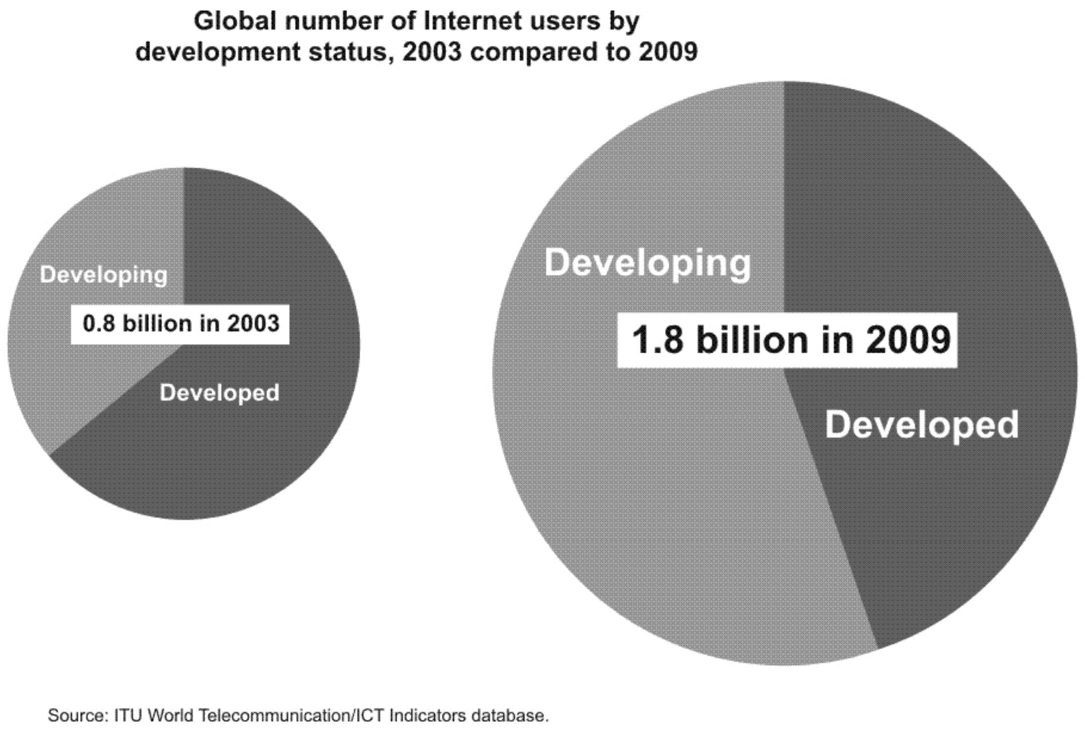


FIGURE 1.1: The growth of Internet, a snapshot[1]

Handley [9] argues as to 'why Internet only just works' with a perspective on the current and medium term challenges faced by the Internet and puts forth a view that a *state of struggle* has been historically the natural state of the Internet which will continue to be so in the future as well. While the strain between the emerging requirements and the existing architecture is expected, there are shortcomings in the current Internet architecture that do not facilitate a positive outcome to the ongoing struggle.

Traditionally, the Internet has been associated with access via fixed nodes. Past decade witnessed the Internet expanding with mobile devices and special characteristic nodes like sensors and vehicles. A perspective of this change can be seen in Fig. 1.2 which indicates the growth pattern in the global ICT domain. The significantly growing trend in the global ICT market place is Mobile device subscriptions and Mobile broadband subscriptions (Fig. 1.3). Putting these trends in perspective of the fact that in 2009, 90% of the world's population now reside in areas covered by mobile cellular signal (Fig. 1.7) indicates that the usage of the Internet is departing from its fixed access roots towards more dynamic mobile access realm.

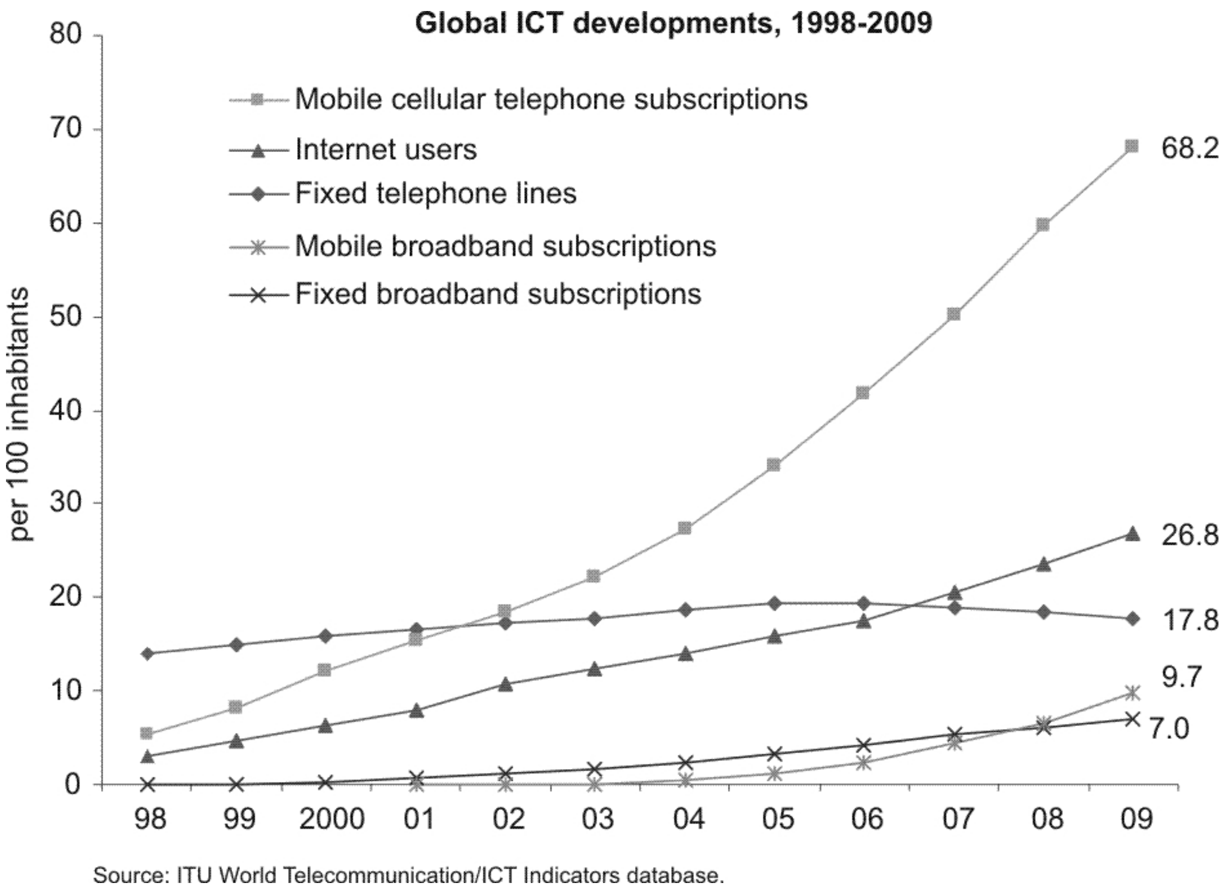


FIGURE 1.2: Global ICT developments (1998-2009)[1]

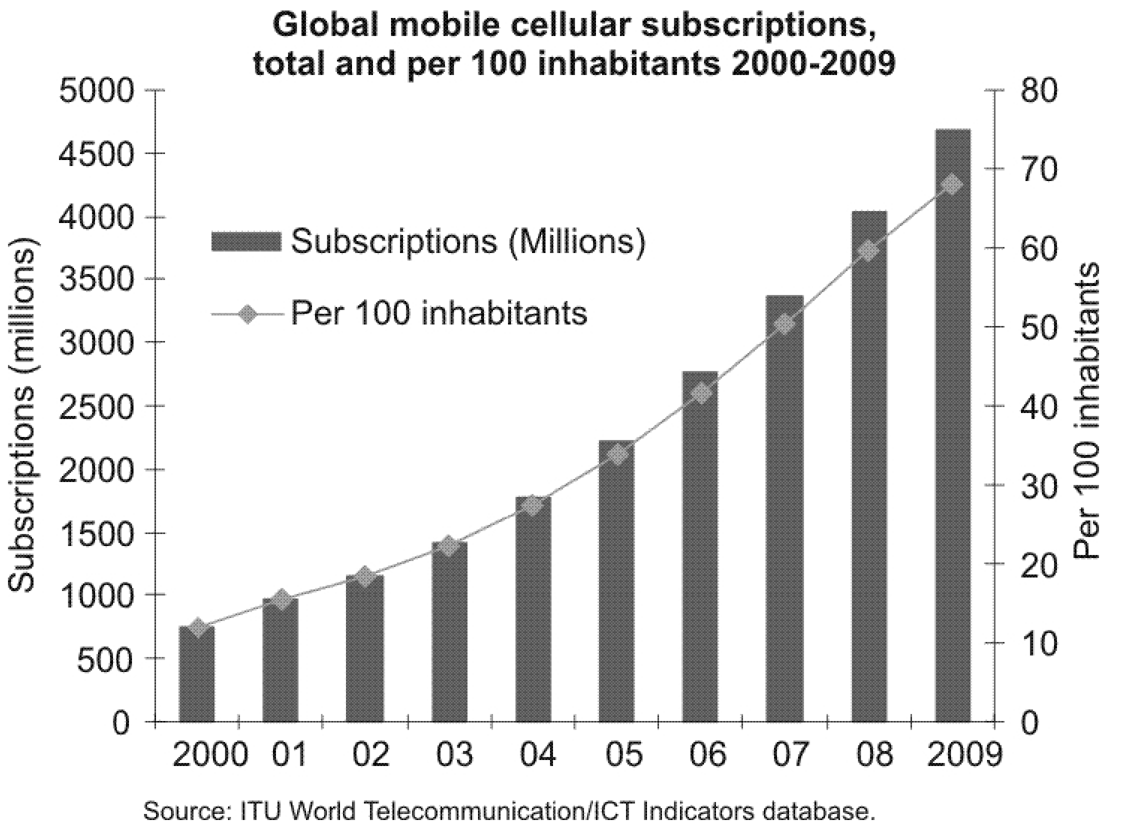
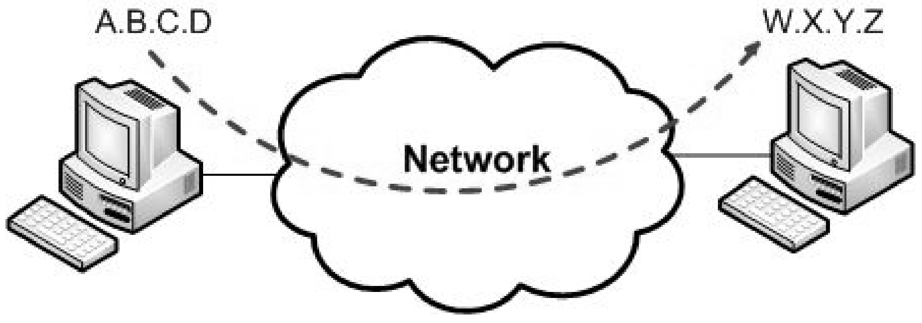


FIGURE 1.3: Mobile telephone subscribers per 100 inhabitants (2000-2009)[1]

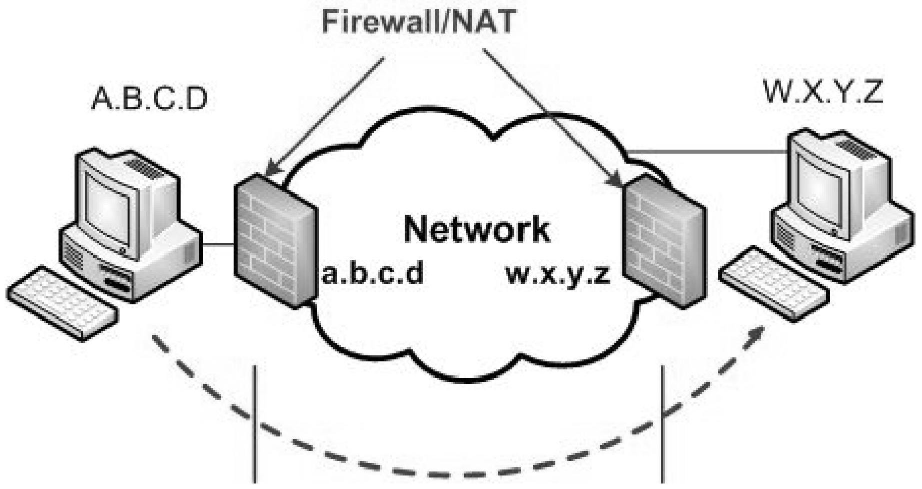
1.1 Motivation and Background

Although far from being efficient and ideal, the design principles of the original Internet are still followed today. The success of the Internet has by itself shackled the possibility of any dramatic change or a completely new architecture from being implemented to accommodate the requirements that were not envisioned in initial design stage. The massive installed base of routers, clients and other network equipment supporting today's network infrastructure make sure that any significant changes to the architecture of the Internet protocol (IP) based network will be overlooked, if not ignored. The financial aspects of migration will play a significant, probably the most important part in migrating to any other replacement proposed from today's architecture. Most of the businesses, governments and academic institutions have invested significant amount of capital into existing network and communication infrastructure, and might resist revolutionary changes which necessitates them to duplicate their investment for similar but a little more efficient functionality.

The current usage of the Internet is at odds with most of the design principles initially conceived. One example can be perceived in Fig. 1.4 where *end to end* principle is violated by middle boxes, NATs, etc. to improve security or accelerate applications using caches [10]. We can also find that *fairness* is restricted via traffic shaping, packet inspection; *best effort* is breached with overlays; *stateless network* concept is infringed by intelligent middle boxes, stateful proxies, label based router, etc [11]. These departures, from the network as a purely transparent carrier of packets, do not follow any new approaches, but are add-ons or overlays piggy-backing on the same old design [10]. These have made the network unnecessarily complex and heavily patched.



(a) End to end Connectivity.



(b) End to end nodes are not visible in the network.

FIGURE 1.4: How Middle-boxes violate end to end principle.

The vastness and distributed control nature of Internet makes it difficult to implement distributed applications with realtime guarantees such as latency, bandwidth and security. Some of the urgent problems like address exhaustion, a uniform mobility model, better compatibility to emerging technologies via header extensions and better security are squeezed into the IPv6; which still addresses only a part of problem, not the network architecture limitations as a whole. There is still a lack of common trust, privacy and security approach, besides inconsistent error handling across communication layers.

1.1.1 Cross Layering: Breaking Layers Intentionally

Ad-hoc networks and sensor networks with millions of nodes pose a massive threat to the traditional internet architecture at various levels [12]. This is not limited to IP address exhaustion, non-routable nodes, limited scope for legacy routing protocols and extreme mobility. The IETF MANET Working Group considers MANETs as an evolution of the Internet, forcing its layered architecture adoption for MANETs. But, as indicated by Godsmith and Wicker [13], a strict layered design is not flexible enough to cope with the dynamic nature of ad hoc networks preventing performance optimizations. Each layer in the protocol stack is designed and operated independently with interfaces between them static and independent of individual network constraints and applications. Security and energy management cannot efficiently be accomplished using the traditional layered architecture as indicated in Fig. 1.5 [2]. The practical approach varies between the extremes of strict layering for compatibility and fully cross-layer design. In such a cross layer design for example, the link layer can adapt rate, power and coding to meet the application requirements given the current channel and network conditions [13]. [1].

[1]Detailed arguments for this approach is discussed in [13]

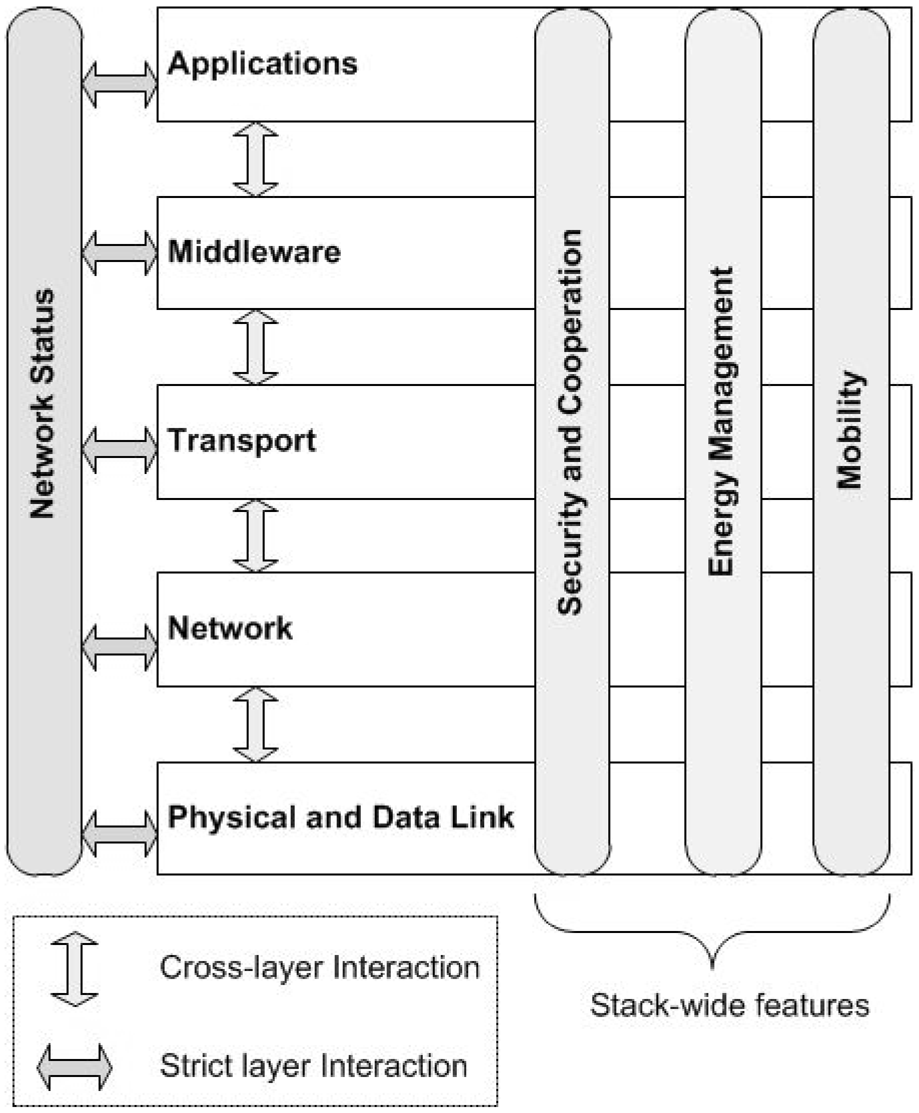


FIGURE 1.5: Cross-layering in MANETS [2]

1.1.2 The Matter of Security

We can find an instance of this complexity in the current Internet security architecture. Security was not as important a concern as openness and fairness during the birth of the ARPAnet. The fact that the network placed no restrictions on connectivity meant that innovative applications could be deployed without obstacles, which essentially lead to the growth of the Internet to the magnitude we witness today. However, the very same design tenet has now made protecting the network from malicious hosts very difficult.

For example, while rudimentary security measures solve most of the problems (e.g., security holes in an applications can be patched and end-to-end security protocols can be deployed, or security overlays for specific protocols), the openness has made it difficult to defend against Denial of Service (DoS) attacks. The security vulnerabilities in the current architecture has been exploited to many nefarious as well as commercial purposes, from cyber warfare [14] to commercial espionage [15] to small scale Denial of Service(DDoS) attacks against individual websites.

As an illustration, Fig. 1.6 (adapted from [3]) indicate the complicated and patched security architecture of the current Internet. The lack of a harmonized security strategy and multiple approaches manifest themselves as cross layering and conflicting overlays.

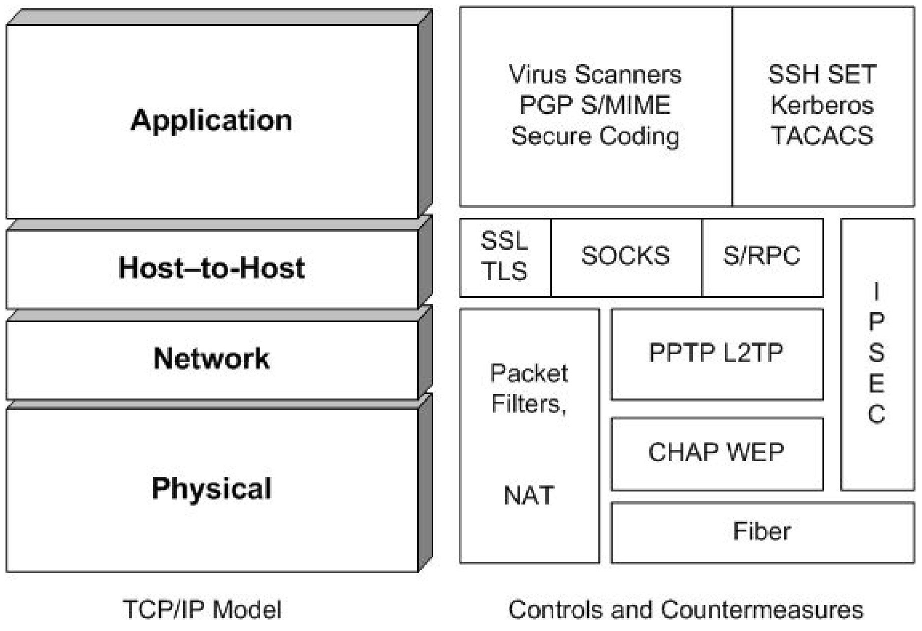


FIGURE 1.6: Internet security controls and countermeasure [3]

1.1.3 Traffic Shaping and QoS

Internet has also affected business models and market places. The basic architectural difference between IP based networks and traditional telecommunication is that data transport is separated from application services. The Internet architecture maximizes innovation in applications and application services. Consequently, network service providers look for innovative ways to profit from the converging or changing market places, by introducing technologies like Deep Packet Inspection (DPI) [16] and content based networking. For traffic shaping and Quality of Service (QoS), routers in the core networks may analyze IEEE 802 frame info, IP header info (TOS bits, port number), TCP header info (receive window) or even the application data stream. This in essence violates the principle of 'end to end' design and fairness of the internet that the network is passive, not to mention the obvious security/privacy implications. Privacy protocols may be used to obscure these fields leading to conflicts and absence of guaranteed or preferential services. Large operators, for example, can negotiate their own peering agreements and disconnect from Internet exchange points, thereby being less bound by 'fairness' rules required to use these exchange points.

1.1.4 Mobility

Mobility as a single critical factor warranting a design review will be inspected in later sections. A brief description of the mobility problem also supports the need for a change in single dimensional layered paradigm. IP decides the next-hop by determining the network information from the destination IP address of the packet (Routing). On the other hand, higher level layers like TCP maintain information about connections that are indexed by a quadruplet containing the IP addresses of both the endpoints and the port numbers. Thus, while trying to support mobility on the Internet under the existing protocol suite, we face with two mutually conflicting requirements:

A mobile node has to change its IP address whenever it changes its point of attachment, so that packets destined to the node are routed correctly. To maintain existing TCP connections, the mobile node has to keep its IP address the same. Changing the IP address will cause the connection to be disrupted and lost as illustrated in Fig. 1.8. Although HTTP redirect and dynamic DNS are used to mitigate this problem, they are insufficient. For instance, if the data changes administrative domains, these solutions no longer works unless the operator of the previous domain provides perpetual support. This challenge cannot be directly solved in today's internet without using multiple addresses or revamping the naming and addressing system.

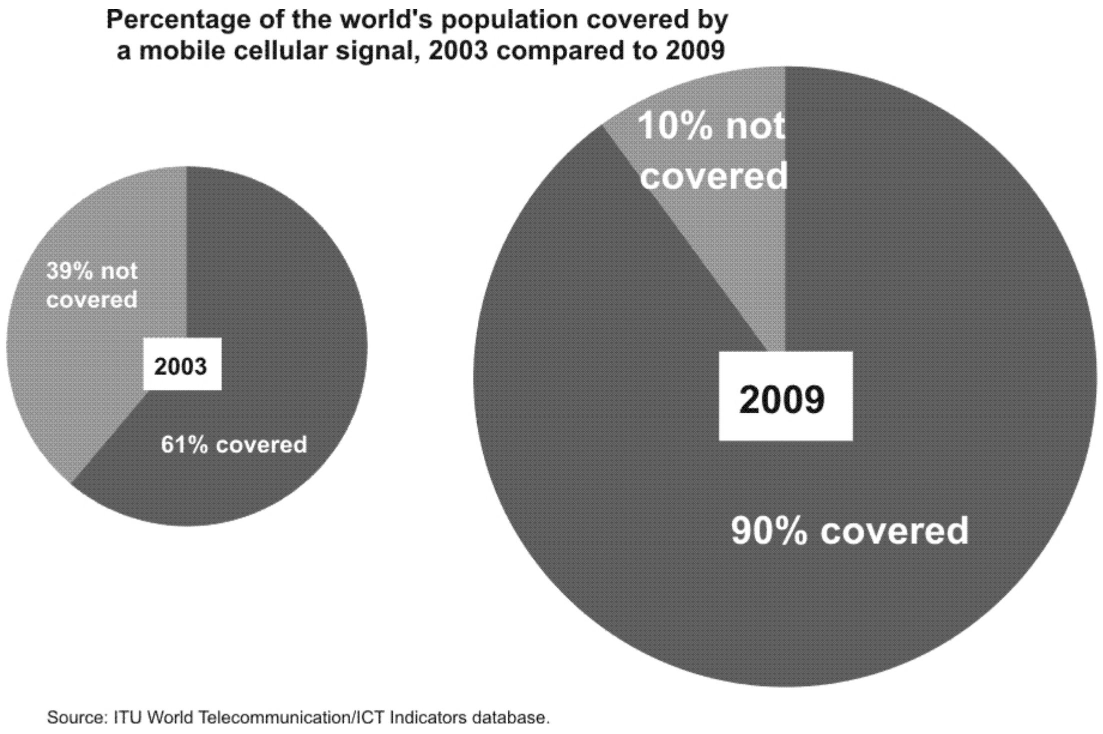
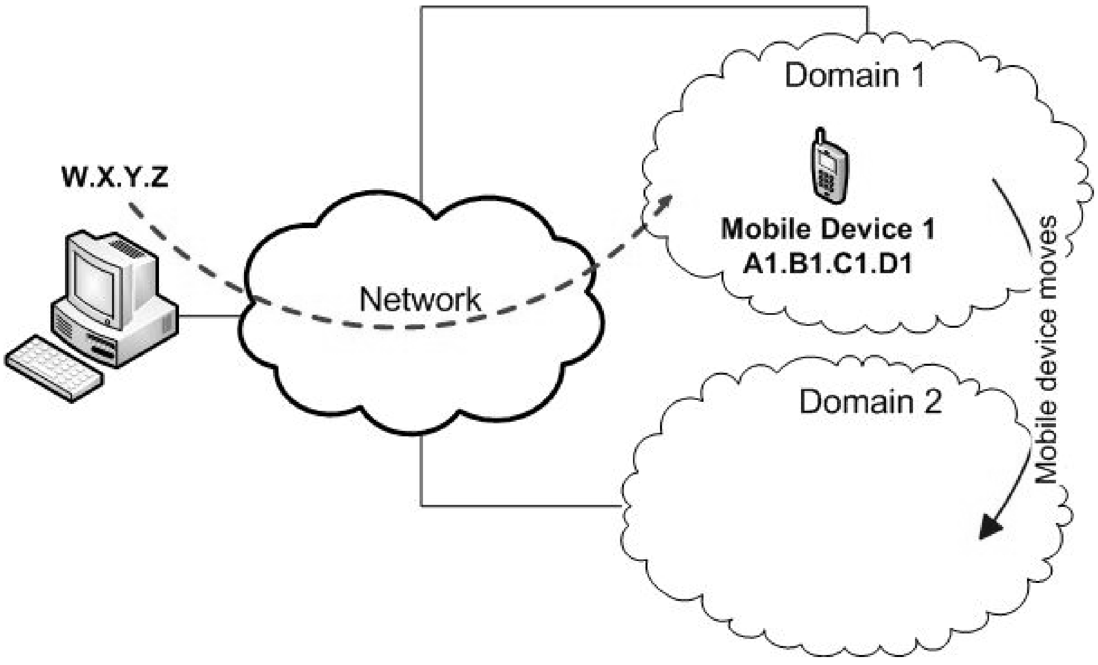
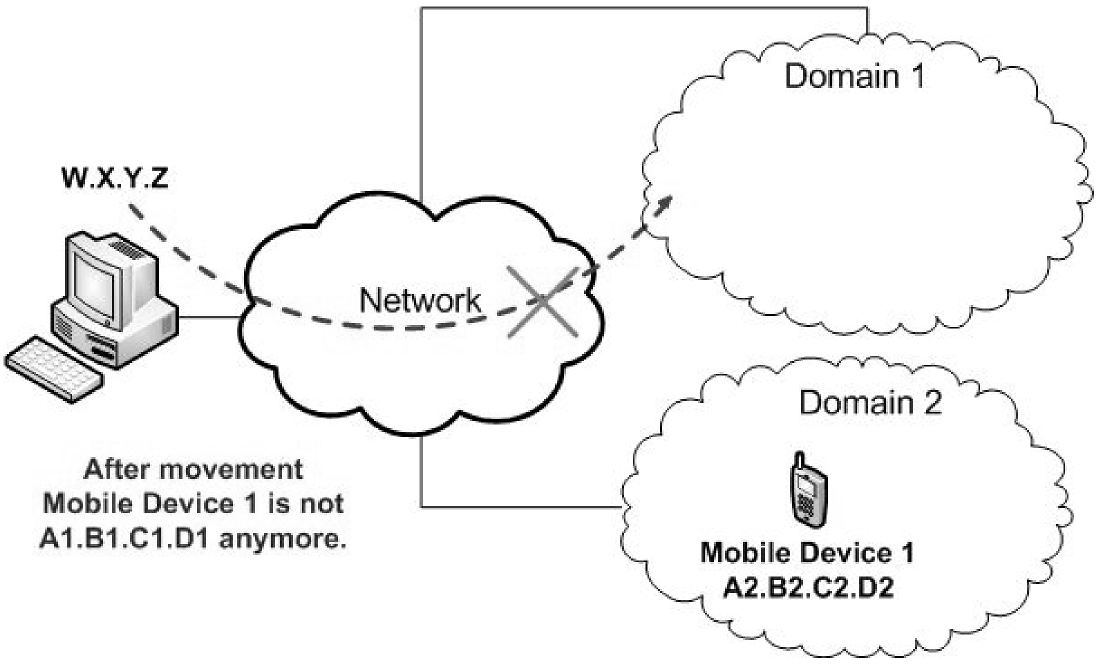


FIGURE 1.7: Cellular coverage (2003-2009)[1]

Addressing the mobility challenge has been the core issue at many research projects. The solutions proposed or work around suggested often tend to work within the limitations of a particular layer belonging to the one dimensional TCP/IP model. The argument on 'which layer does mobility belong?' has even been studied with varying outcomes [17, 18]. We believe that mobility, like security, is not a layer feature but an aspect that characterizes an architecture. This means that it cannot be implemented efficiently by



(a) Mobile Device 1 changes location.



(b) Mobile Device 1 renews address making it unreachable.

FIGURE 1.8: How mobility breaks naming and addressing.

tweaking a single layer or even layers, since the interface among the layers being still static will beat the purpose of cross layer cooperation anyways.

1.1.5 Cloud Computing and Scalability

Over the past few years, the term *Cloud computing* is coined to present both a model and new opportunities for technology buyers in an enterprise [19, 20]. The essential characteristics of cloud computing include both the delivery model and the commercial model. In terms of the delivery model, it should be available as a service over the Network (most commonly, the Internet) and accessible either from a Web browser or as a Web service. Commercially, users pay for service usage; both the overall maintenance effort and user costs are low. The attractiveness of Services available in the network lies with the opportunity for enterprises and businesses to focus on the core capabilities while outsourcing certain aspects of required IT Services at marginal cost. The support hassles are transferred to the cloud-service provider, while accelerating provisioning and deployment. The Cloud Computing paradigm, while hyped to an extend [21], has the capability to alter the way in which organizations build their infrastructure and applications.

Various standards and technologies such as server virtualization, Web Security, and Web Services come together in implementing Cloud Computing. Cloud Computing models could offer, among other, Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS) [22, 23]. Cloud Computing adoption exports security, performance, availability and reliability concerns from enterprises to Cloud Service Providers, who have to address these issues through appropriate architectural measures and service-level agreements (SLAs) to gain user confidence in the Cloud.

There are certain requirements that need to be in place before Cloud Computing becomes the norm. One is the clear definitions of domain boundaries. Interconnecting services across domain boundaries means the need for procedures to establish and monitor service *relations* across boundaries. With multiple Cloud Service Providers available to choose from, robust service discovery and negotiation facilities (preferably automated) need to be in place. There is also the need for standardized commonly adopted *negotiation* and *meta-negotiation* languages [24, 25].

1.1.6 New Paradigms

The Internet evolution has been characterized by ingenuity on the part of software and application designers to circumvent the architectural limitations, and has brought into play varying *players* into the sphere of influence [7]. The resulting *tussle* influences not only the direction of the future evolution of the Internet, but also the nature of the next generation architecture from the notions on design along tussle boundaries, leave space for tussle to play out, etc. However, it is worth inspecting the origin of some of the new requirements, as it can help us classify common trends into parameters or independent modules in a future architecture. In Fig. 1.9, we show some of the new requirements and possible motivations for architectural changes.

Technologies emerge independent of the Internet, which in due course requires interacting with the network for various reasons. We find powerful mobile communications devices, sensors, medical implants, vehicles and a host of other network capable appliances emerging. Besides, new networking models like ad-hoc networks, vehicular and sensor networks present challenges which are not efficiently handled by the current architecture [26, 27]. The number of connected nodes in the Internet has gone from a few in the early eighties to millions (currently), with a strong possibility to be trillions [28] with the inclusion of ad-hoc nodes, cheap sensors and networked vehicles in the future.

A change in the programming domain reflects the change in the nature of applications being implemented on the Internet. As programming frameworks migrated from functional to object oriented, Internet evolved from a single application network to become distributed and service-oriented. User demands like interactive resources, user generated content, content sharing, local access of distributed content, anywhere/anytime access of own data, etc. requires the underlying architecture to support a set of basic capabilities, which are inefficiently implemented into the legacy Internet [29]. The demands can also include demands based on behalf of the users by other entities such as governments, corporations and content owners. For example, the open and end-to-end nature of the

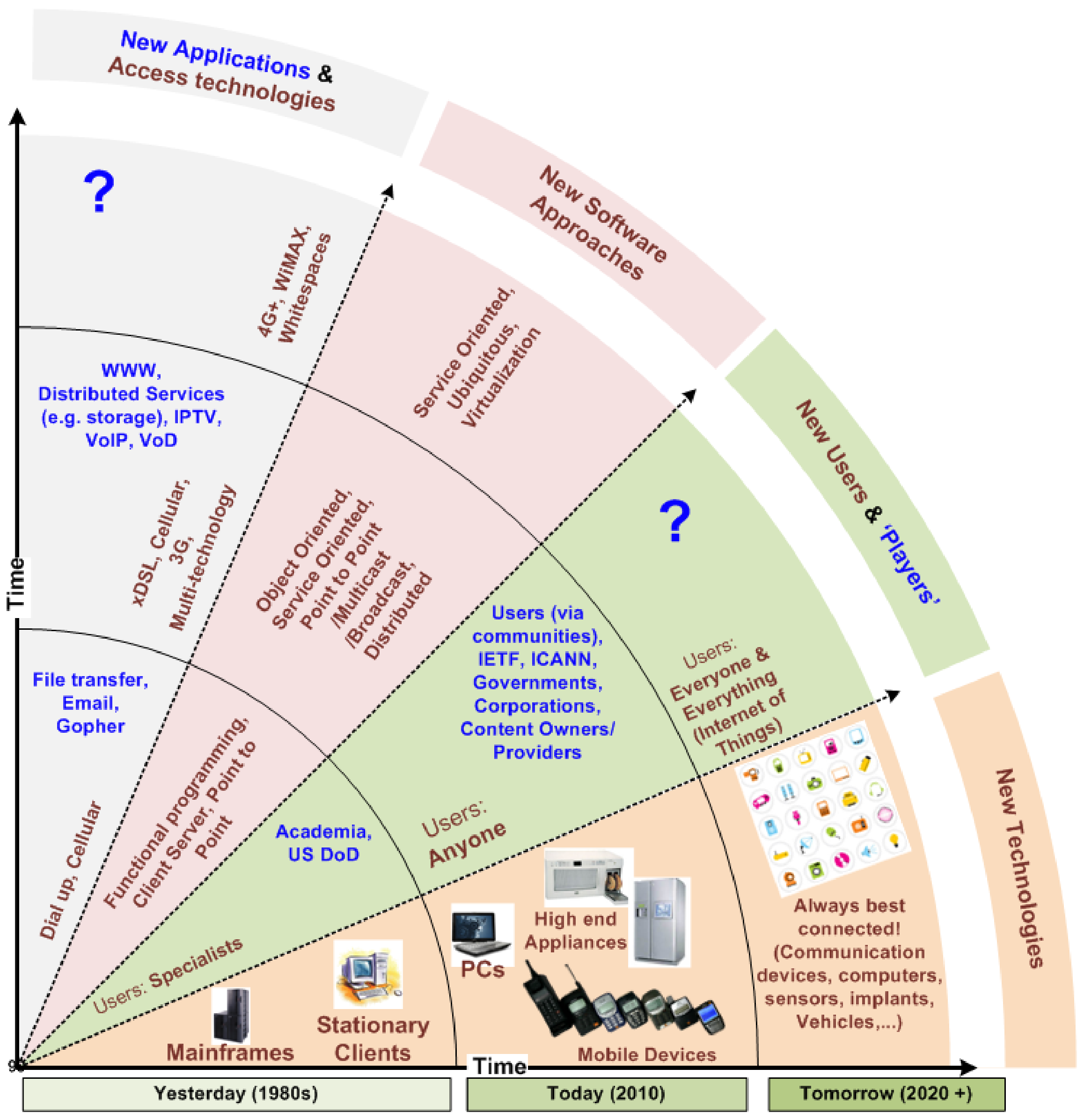

New Applications & Access technologies
New Software Approaches
New Users & 'Players'
New Technologies
?
4G+, WiMAX, Whitespaces
Service Oriented, Ubiquitous, Virtualization
WWW, Distributed Services (e.g. storage), IPTV, VoIP, VoD
xDSL, Cellular, 3G, Multi-technology
Object Oriented, Service Oriented, Point to Point /Multicast /Broadcast, Distributed
?
Users (via communities), IETF, ICANN, Governments, Corporations, Content Owners/ Providers
Users: Everyone & Everything (Internet of Things)
File transfer, Email, Gopher
Dial up, Cellular
Functional programming, Client Server, Point to Point
Academia, US DoD
Users: Anyone
Users: Specialists
Mainframes
Stationary Clients
PCs
High end Appliances
Mobile Devices
Always best connected! (Communication devices, computers, sensors, implants, Vehicles,...)
Time
Time
Yesterday (1980s)
Today (2010)
Tomorrow (2020 +)

FIGURE 1.9: A summary of Internet evolution

Internet is broken by middle boxes to accommodate for address exhaustion, security etc [10]. This necessitates architectural changes incorporating such additions [30].

These are not independent driving forces. These factors tend to influence each other to a stage where the underlying architecture can no longer efficiently support the newly construed paradigms. This will be the case with any new tightly designed architecture. The boundaries of such architectures will be tested. One of the more accommodating architectures would be the ones which account for this growth ('design for tussle', for example [7]), or a system modular enough to keep pace with the innovations around it. The idea of service oriented architecture and service composition, manifested in different proposals as network composition (e.g. Ambient Networks) must be considered in this context.

1.1.7 Adaptability, Migration and Evolution

Once it is clear that critical problems exist with the current architecture and a new thought process is required from an architectural perspective, there are two common methods of approach that can be utilized. One is the ‘Incremental Approach’ to try fixing some immediate and pressing problems. The other approach is to have a ‘Clean Slate Thinking’ to fix all the problems that can be identified as being inherent in the current architecture. A study of the current state of the art will reveal that both approaches are being explored by various entities globally, substantiating the necessity and urgency of such a transition [30–32].

An inspection of the life cycle of similar attempts can provide critical input into the accommodating nature or the resilience of current implementations and business models. Architectural changes to the core of the Internet (E.g. IPv6) and add-on/overlay services (E.g.: MIPv4, MIPv6, IMS etc.) have met with varying levels of success. There are a host of new architecturally superior implementations and changes dismissed *a priori* by the marketplace, due to reasons such as ossification of the TCP/IP model and financial aspects of bringing about a dramatic change in the currently installed infrastructure base. It is easier for researchers to consider a clean slate approach of a new architecture, protocols and service. However, such an approach is generally unacceptable due the changes required to already existing infrastructure and devices. This approach, while ideal from a research standpoint is difficult to implement in the current scenario.

The incremental approach to addressing the current limitations is attractive to service providers and network operators in terms of cost and availability. This approach, however, produces inefficient and often complicated solutions. An ideal solution to this conundrum will be to suggest modular incremental changes to current architecture aimed at addressing immediate problems, which functions as a milestone or part of the transition towards a completely reconsidered and modular network architecture. The idea of overlay networks (for example, Peer-to-peer overlay networks [33]) is quite relevant in this context, where a new technology can be implemented at 'present time' over existing architecture, so as to bring in the new functionality without a radical change to the underlying architecture. This functionality, at a later time can be accommodated as a part of the architecture itself, if designed to relevant open standards.

The above mentioned approach is not only true for functionality overlays like IMS, but completely reworked architecture as well. Consider the migration towards IPv6 from IPv4 in this context. The newer Internet layer (IPv6) can coexist with the current one (IPv4) via gateways connecting islands of IPv6 routers to the IPv4 world, software encapsulation like the 6to4 transition mechanism [11] or tunnelling etc. [34]. In the future, when most of the nodes (routers, specifically in this case) support IPv6 in the future, the IPv6 'islands' automatically become the 'main network', with IPv4 becoming 'legacy islands' interfaced via 'legacy' gateways. This approach however requires a large scale consensus and collaboration from major players in the research community and industry.

1.1.8 Other Challenges

The lack of a common trust, privacy and security approach is just some of the shortcomings of the current Internet architecture. The Ambient Networks project, a European sixth frame work project identifies some of the requirements to be addressed for their next generation communication architecture [35]. Haggle [36] identifies that the root cause for some of the usability deficiencies with regards to mobile devices today arises from the synchronous IP-based APIs presented to applications along with the numeric addresses as end-points. Applications implemented in such models rely on networking infrastructure for end to end communication without taking advantage or being aware of local or neighboring resources. Besides, any *unconventional* usage or resources force users to possess high technical awareness of their connectivity environment. We can list a few of the challenges or requirements encountered by applications as follows:

- Concept of Location/Neighbourhood awareness, proximity etc.
- End to end service oriented communication.
- Separation of identifier and location in naming and addressing.
- Session continuity & management across domains.
- Common trust, anonymity and federated identity management.
- Parameter/Metric based routing (added value based routing).
- Routing facilities based on application layer needs.
- Multi homing, delegation, indirection.
- Capability signalling across devices, domains.
- Real-time and Distributed real-time application requirements like priority, guarantees etc.
- Scalability for trillions of nodes.

As can be observed from the above list, the issues to be addressed are rather basic and spread across the existing ‘layers’ of TCP/IP model; i.e., it is difficult to solve the above shortcomings at a particular ‘layer’ of the current Internet architecture. The stress on the current architecture is not limited to the new usages of existing technologies, but also arises from new technologies that are incompatible, but forced into compatibility for legacy interoperability. For instance, Ad-hoc, vehicular and sensor networks differ dramatically from the relatively static ‘client-gateway-server’ design of the Internet with the number of nodes stretching into billions and extremely dynamic mobility scenarios.

A migration of applications, services and devices (voice communication, video distribution) from traditional mode of communication to IP based networks have further complicated the network as well as the business models. Different overlay networks and addons to the traditional architecture handle the architectural necessities of such requirements, albeit inefficiently. Thus, Internet’s increasing ubiquity and prominence have made its flaws all the more apparent and addressing them, urgent. There is a consensus that the architecture of the internet needs revisiting [37–42].

1.2 Organization of Thesis

The remainder of this thesis is organized as follows. The state of the art in current approaches that are being investigated to overcome the shortcomings in the existing Internet architecture is covered in chapter 2. In chapter 3, we discuss the various aspects of Service Oriented design aimed at communication and networking architectures. We introduce the basics of Services and explore the various Service design principles as well. The approaches used to combine various Services to form a more useful composite Services are further discussed together with the challenges faced during the process.

In chapter 4, we propose the principles supporting our architecture and formalize the new Service Oriented network architecture. We also demonstrate two instances of such an architecture. We discuss the proposed architecture in chapter 5 together with the benefits of our approach and contrasts them with the shortcoming. We compare our approach with few of the most relevant competing proposals and finally concludes in chapter 6, proposing future work to move the study forward.

1.3 Summary

The table below summaries the issues discussed in this chapter and sets the goal for this work.

INTERNET DEFICIT/ISSUE	GOAL OF THE THESIS
Broken end-to-end Principle	Propose a Service to Service paradigm
Inefficient Layering	Propose Service Oriented layer-less architecture
Fragmented Security Solution	Integrate security into the architecture
Lack of uniform Mobility and Session stability	Handle session dynamics in a flexible manner
Uncertain Scalability	Provide aggregation capabilities to aid scalability
Cumbersome Evolution / Adaptability	Separation of Implementation and Interface
Inefficient Virtualization support	Accommodate virtualization features
Difficulty in Integrating Business borders and policies	Recognize business borders
Inefficient Real-Time support	Delayed binding and QoS Awareness within the architecture

CHAPTER 2

STATE OF THE ART

Before developing a new architecture we study efforts to overcome shortcomings of the current Internet in the first part of this chapter following a bottom-up approach. The second part focuses on service driven architectures that try to change the Internet from a top-down approach. This thesis has components of both these solutions to have a realistic chance to contribute to the ongoing discussion on the direction the Internet evolution.

The original design tenets for the internet such as *self describing datagram*[1], *fate sharing*[2], *layered abstraction*[3] and *end-to-end principle*[4] [43] were relevant for early static packet based best effort communication paradigm and some of them are relevant still. But as more and more new requirements come into picture, the old architecture strains to

[1]Control information should be carried within each datagram including necessary address and identifier information necessary to route datagrams between end points

[2]State information should be used within the entity where it is used

[3]A hierarchical protocol structure with a clear identification and separation of services provided by each layer, with networking layer as the unifying layer supporting heterogeneity of higher and lower layers

[4]The principle limiting the implementation of functionality in a network

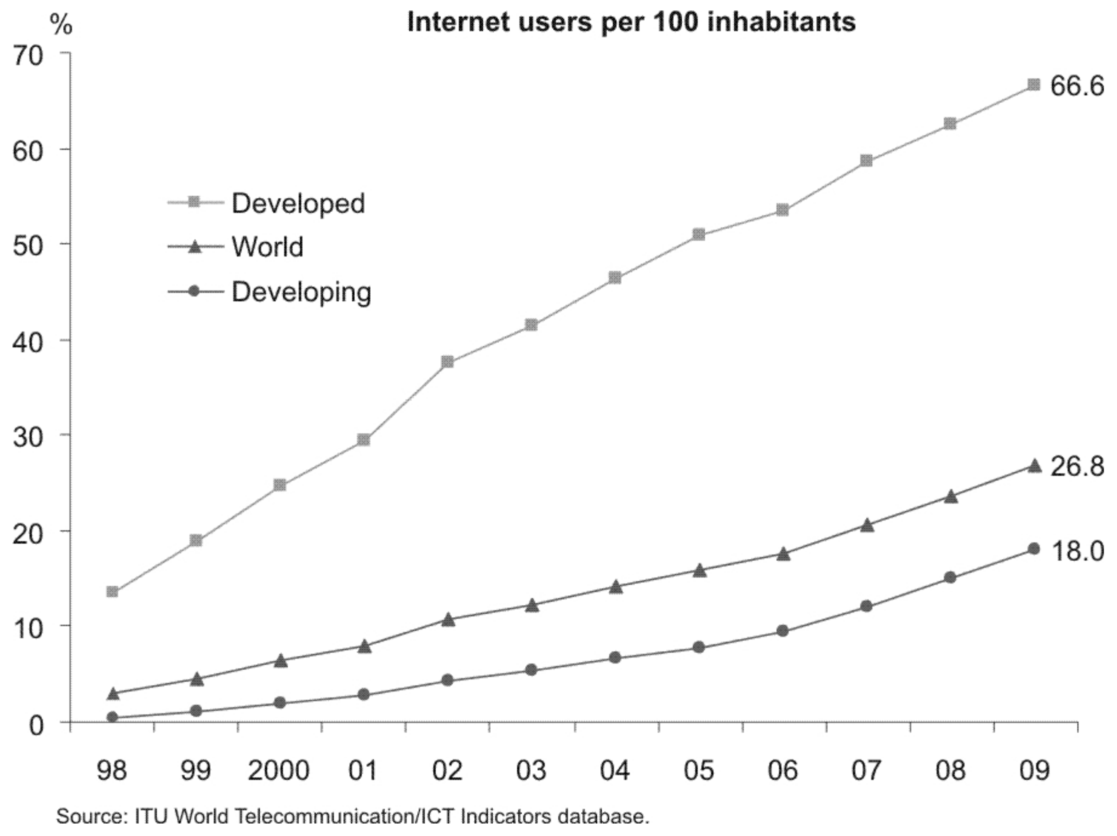


FIGURE 2.1: Internet users per 100 inhabitants (1997-2009)

accommodate them. Engineering solutions are proposed to address these new requirements and these optimizations essentially break the original design principles. In the sections below, we discuss some proposed solutions and state of the art for the problems suggested in section 1.1. This chapter also highlights the effort that networking community put into engineering solution to circumvent the limitations of the architecture, leading to a less elegant overall implementation. Subsequential to the initial design, most crucial redesigns (like DNS, Link state routing protocols, BGP) to the network were undertaken to mitigate scalability issues [9].

2.1 Current Efforts to Overcome Internet Network Architecture Shortcomings

While the internet has tried to avoid vertical silo (or smoke stack/chimney model) effect by abstracting horizontally with layers rather than complete end to end solutions, in reality, the effect has been more hard-coupled. The proliferation of IP as the *de facto* standard for network abstraction has made the generic concept into an IP-Hourglass model (Fig. 2.2) [44]. This model has worked remarkably well over the past few decades, but does not perform well when exercised against the new paradigms and applications. There is an interesting discussion growing around the 'waistline' of the internet with various opinions and concepts emerging on 'what will or should' replace or added to expand the waistline of the current network architecture [32, 44].

With the shortcomings of IP (such as the identifier/locator dichotomy, inability to accommodate multihoming) [45], it is only logical to provide an alternate addressing scheme to take advantage of the innovation in the networking and routing domains over the past few years. However, '*what is the best alternative?*' is still an open question. It is this openness that should be embraced rather than providing a solution which in a few years will find itself inefficient or even unsuitable for use due to newer unforseen requirements. The self contained packet and the best effort delivery makes no differentiation of control and data flow in IP-Datagrams. Control of the data delivery is becoming more relevant as applications require more than just best effort guarantees. Certain protocols like Session Initiation Protocol (SIP) emerged to incorporate a concept of session and flow management. The emergence of overlay architectures over IP networks also attempts to addresses this shortcoming.

In a best effort packet delivery mechanism like IP, the concept of a separate control channel is diminished (apart from already existing internal and external routing protocols) [44]. Each packet carried enough information for processing at intermediary nodes. Following convergence and adoption of IP as the *de facto* abstraction at the

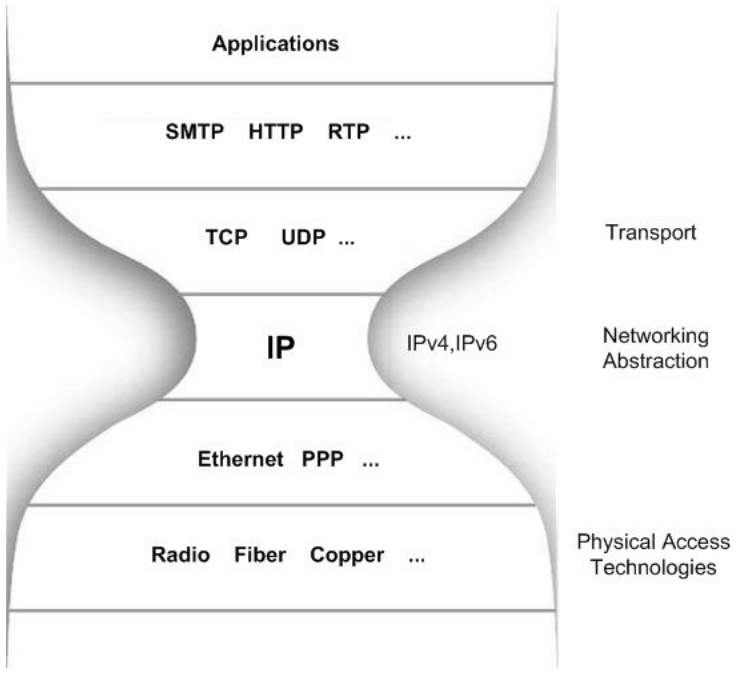


FIGURE 2.2: The IP Hourglass illustration

network layer, the concept of the self contained packet became the norm. This simple and common layer was advantageous for interoperability but is manifesting as a major challenge for the flexibility for applications running at higher layers. With the new requirements like QoS and security, extra intelligence needed to be built into routers to examine the contents of each packets to decipher what needed to be done with it for additional functionality (as per end user signalling through RSVP, for example). With the emergence of realtime multimedia as a major part of the traffic on the internet, separate control protocols had to be developed (SIP, RTP) just to facilitate the delivery of multimedia streams/sessions. With QoS, routing is no longer simple forwarding of packets but consists of prioritizing, queuing, dropping, tagging/marking and so on. The routing architecture of the current Internet does not support packet forwarding based on such rich or descriptive parameters.

These trends segment the Internet as a collection of application level networks (BitTorrent, IMS etc), each overlay addressing a specific application requirement or functionality. Different control protocols implemented in a distributed fashion decide the nature of the overlays. As articulated by Aguiar [44], the concept of in-band control signalling through the packets necessitates the ‘hard’ processing of each packet to provide additional functionality to the flow. This is hardly efficient when the choices become numerous and the packet count follows suit. A separate control plane or architecture might be necessary, independent of data flow to facilitate added functionality to the communication via networks. This will facilitate capability negotiation across different control domains (like businesses, Autonomous Systems etc) for setting up sessions or temporary peering agreements separate from data delivery mechanisms. The proposal to have an additional hourglass model for the control architecture complementing the data hourglass model (with IP at the waist) [44] for networking might be one approach to address this challenge. The idea of separation of concerns (control and data) within the network architecture brings flexibility at the cost of simplicity.

2.1.1 Cross-Layering in Mobile Ad-hoc Networks

The layered architecture has its limitations. A strict layered design cannot cope with dynamic environments like Mobile ad-hoc networks (MANETS) , preventing performance optimizations [2]. For instance, the mutually independent implementation of multimedia streaming and compression algorithms at application layer without considering the resource management, adaptation strategies, error protection or scheduling available at the lower (physical, MAC or network) layers leads of inefficient usage of resources. A couple of work-arounds to this inefficiency have been proposed in *Layer Triggering* and *Cross-Layer Design* [12, 46, 47], specifically where performance related optimizations are significant like wireless sensor and ad-hoc networks. Since wireless links cerate problems for protocol design that cannot be handled well within the framework of layered design, network designers were motivated to violate the layers [46]. This approach allow applications to adapt based on underlying channel and network characteristics and *vice versa*. To enable such interactions, the information in layers is shared across layer boundaries.

However, most of the implementations still fall short when it comes to adapting to QoS requirements, dynamic channel conditions and interoperability across heterogenous networks [48]. Kawadia and Kumar [49] cautions against applying too much cross-layering especially optimize wireless network performance and concludes that such an approach 'can run at cross purposes with sound and longer term architectural principles, and can lead to various negative consequences'. This is relevant as an unbridled cross-layer design can produce 'spaghetti-like' design that is impossible to maintain efficiently. Every modification must be propagated across all protocols and can produce unintended interactions among protocols, such as adaptation loops, that result in performance degradation. We conclude that such as approach is not in the interest of a long term evolution of a solid and stable network. The longevity and proliferation on the Internet was based on its sound layered architecture, and modifying this design for performance optimizations via violating the design principle is unsustainable. While the above cases are made mostly for the Network layer and below in the OSI architecture, the cross-layer approach is affects higher layers. For example, Common transport protocols (like TCP and SCTP) also includes network layer state. FTP uses network layer information (IP address) directly on the FTP control channel.

2.1.2 Naming and Addressing

The issue of separation of naming (identifier) and location locator for routing has been a recurring theme in network architecture evolution discussions, for example, with in the Name Space Research Group (NSRG)[50] within the IRTF. Under the current network architecture, the applications are responsible for setting up all bindings required for communication. This necessitates that the software is written to specific underlying network architecture, without modularity. The close binding also makes it difficult for developers to implement applications and solutions that can adapt to new communication mechanisms. Most of the new suggestions, proposed to overcome such limitations concentrate on abstracting the applications from the underlying network architecture, mostly by adding one more abstraction layer over the existing naming system (using IP addresses to identify and locate end nodes). These approaches, to an extend, mitigate the ill effects from current dual usage of IP addresses as end point identifier (name) and location (address). The reliance of certain applications on the Domain Naming System (email, web addresses etc.) together with the inability of the DNS to adapt to rapid updates make it more difficult in dynamic mobile environments.

The newly adopted (albeit slowly) IP version 6 (IPv6) tries to mitigate most glaring problems related to IPv4 like address exhaustion, ownership allocation, scaling and mobility (partially). IPv6 provides a substantially larger IP address space that IPv4 and does away with the requirement for NATs[1]. But, the basis for IPv6 follows the same arguments as IPv4 and inherits the latter's shortcoming due to the requirement for some backward compatibility.

Host Identity Protocol (HIP) [51] identifies the naming and addressing of entities as the key challenge in today's architecture and proposes as a solution to separate them, decoupling the usage of the address (i.e. the IP address) as the identity of resources or nodes. The separation is achieved by introducing a new layer between the conventional TCP/IP stack between the network layer and the transport layer. HIP uses cryptographic identifiers as the namespace which helps to integrate baseline end-to-end security into the architecture when used with Diffie-Hellman [52] and appropriate security protocol, such as Encapsulated Security Payload (ESP) [53]. The layered naming architecture [30] builds on the concept of HIP to propose four layered naming abstraction with three levels of name resolution in between: user level descriptors, service identifiers and endpoint identifiers and finally IP addresses (or other locators) to name network locations.

Other suggestions such as '8+8' addressing architecture [54] and SHIM6 [55] are more of engineering solutions instead of or based on IPv6 addressing schemes. Ambient Networks Project (AN) [56], part of the European Union's Sixth Framework Program [57]

[1]This feature makes NAT transversal for applications unnecessary and better support for peer-to-peer applications

proposes to use naming architecture which adopts a layered naming model, with separation concepts borrowed from layered naming architecture and HIP [30, 51]. Intentional Naming System (INS) [58] proposes a simple language based on attributes and values to be used as names. Applications use the language to describe what they are looking for (intent), not location (or hostnames), and INS incorporates resolvers which self-configure to form an application-level overlay network to discover and bind to end nodes.

To accommodate heterogeneity across networks, the Web-services Addressing work group proposes the WS-Addressing as transport-neutral mechanisms to address Web-Services and messages [59]. This specification enables messaging systems to support message transmission through networks that include intermediate processing nodes such as enpoint managers, firewalls, gateways etc in a transport neutral manner. WS-Addressing defines two interoperable constructs that convey information typically provided by transport protocols and messaging systems. These constructs, *endpoint references* and *message information headers*, normalize this underlying information into a uniform format that can be processed independently of transport or application. The endpoint references (built on URI) convey the information needed to identify/reference a Web service endpoint[1] and also to provide addresses for individual messages sent to and from Web services. Message Information headers convey end-to-end message characteristics including addressing for source and destination endpoints as well as message identity.

2.1.3 Security

Security in networks has been a much studied realm within ICT research with shifting focus from information assurance to network infrastructure security [60]. But as discussed before, security by itself was not a significant design tenet for the Internet architecture. Under IPv4, it is the responsibility of the email client at the end nodes to provide those

[1]A Web service endpoint refers to an entity, processor, or resource where Web service messages can be targeted.

services. Today, the Internet faces threats such as Denial of Service attacks, malicious code distribution, man-in-the-middle (MIM) attacks, fragmentation attacks and reconnaissance attacks [14, 15].

Introducing middle-boxes like Network Address Translation (NAT) and Network Address Port Translation (NAPT) provided some level of protection against some of the threats mentioned above using methods such as firewalls. The introduction of the IPSec protocol, allowed some communication to be encrypted but its implementation in IPv4 is optional and the whole responsibility of ensuring secure communication still lies with the end nodes. Such security measures are too little and too fragmented to provide comprehensive security, especially to new applications like mobile e-commerce and portals which demand end-to-end security.

In IPv6 [61], IPSec as a mandatory major protocol requirement ensures better security than IPv4. Besides, IPv6 also mandates cryptographic extensions. IPSec contains a set of cryptographic protocols for ensuring secure data communication and key exchange. The main protocols used within IPv6 include Authentication Header (AH) protocol, which enables authentication and integrity of data, Encapsulating Security Payload (ESP) protocol [53], which enables both authentication and integrity of data as well as privacy of data and Internet Key Exchange (IKE) protocol [62]. The latter protocol suite helps the initial set up and negotiation of security parameters between two end points, and subsequent communication security over the lifetime of the session. These end-to-end security mechanisms within IPv6 provide authentication and encryption abilities to all applications thereby eliminating the need for applications to have integrated support for such abilities themselves.

2.1.4 Mobility

Many engineering solutions have been proposed to accommodate a mobility paradigm within the Internet architecture designed for static nodes. The pace at which mobile devices with Internet access on the move is growing at an exponential rate (Fig. 2.1). Since IP is the most widespread unifying layer, IP mobility has been the most researched in terms of solutions for device mobility. However, due to the limitations with the IP network architecture and routing architecture it is difficult to attain true mobility using a IP single address. Mobile IPv4 [63] is one of the proposed solutions for mobility support in IPv4 networks. Each mobile node is assigned a home address (the address of a home agent). When a mobile node is not at home, it conveys information about its present location, also called, care-of-address to the home agent. If a node wants to communicate with this mobile mode, it will first send the information packets to the home address, which in turn forwards these packets to the care-of-address of the mobile node. However, this solution is not optimal as it maps the telecom solution for subscriber mobility into the packet based networks without the support of a separate control channel. Besides, using the IPv4 address for transport layer sessions (home address) and using a different IP address for routing packets to the end node is not a uniform solution. Other shortcomings of this solution include the requirement for a special router in the location of the mobile node for the node to be mobile, lack of mandatory route optimization and ingress filtering problem [1].

Mobile IPv6 (MIPv6) [64] indicates the mobility solution corresponding to IPv6, where mobility support is mandatory. Route optimization is also a mandatory feature for MIPv6. The need foe a special router at the visiting network is avoided using features like Neighbor Discovery and Address Auto-configuration. Ingress-filtering problem in MIPv6 is avoided through the use of care-of address as the source address. IPv6 also has a large address space ensuring that each mobile device can have its own unique IPv6 address.

Ratola [18] the mobility solutions existing at different network layers - MIPv6 at layer 3, HIP at layer '3.5' and SCTP at layer 4. He admits that there is no straight forward solution to the choice of a layer for mobility. Each traditional layer is either overloaded

[1]The correspondent node uses the home address as the source address of the packet and there may be confusion on which IP addresses it should be allowed to accept or not

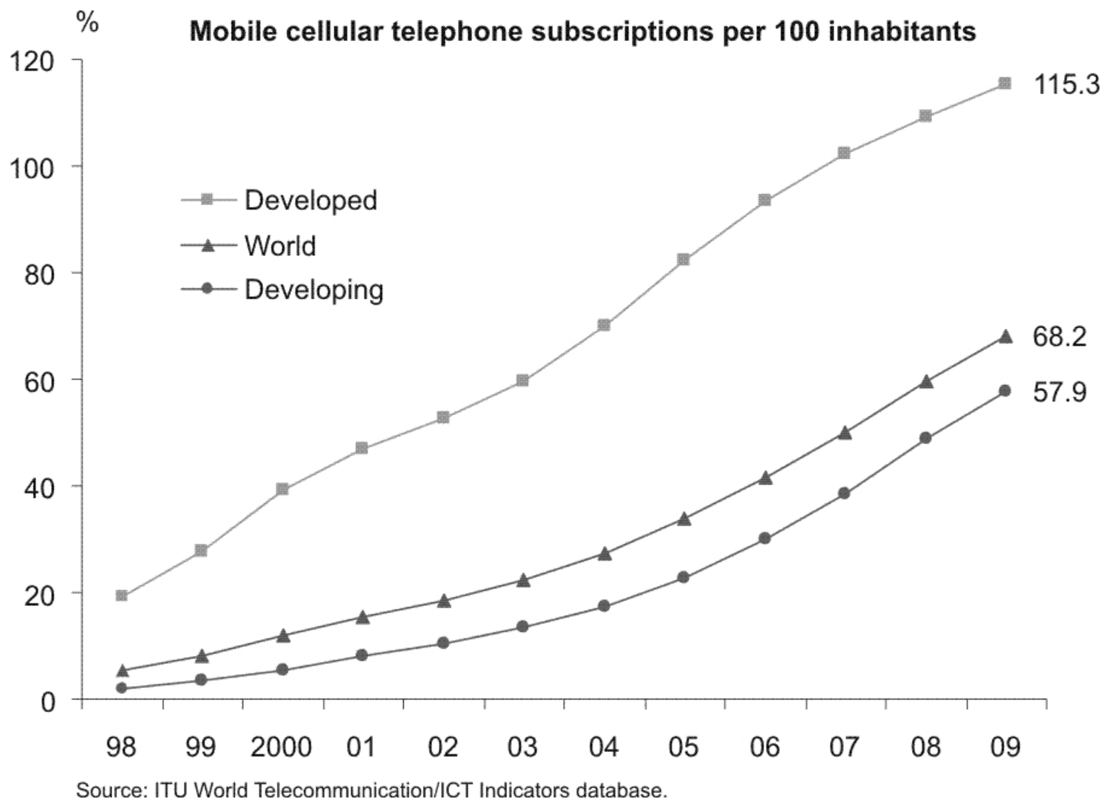


FIGURE 2.3: Mobile telephone subscribers per 100 inhabitants (1997-2009)[1]

with functionality or lacks flexibility to to implement a comprehensive mobility solution within the layer boundaries. The HIP mobility solution seems to the better among the compared ones, but the ultimate conclusion, as expected, is ‘to rethink and renovate the whole architecture’ as some modification is anyway necessary for implementing HIP based mobility solutions.

The current need for engineering solutions and optimization in mobility protocols is indicative of the architectural limitations with in the current approach. Other proposals for architectural modifications includes new *tussle aware* design principles such as information exposure, separation of policy and mechanism, fuzzy end principle and resource pooling [65]. Several solutions [66–68] have been proposed to accommodate mobility and Multihoming for alternative proposal using HIP.

2.1.5 Scalability

Scalability has always been an issue with the Internet architecture. In fact, most of the changes to the core of the network (DNS, CIDR, BGP) and the current ongoing changes being applied (migration to IPv6) can be partially attributed to scalability concerns. With the rapid growth of routing tables, Classless Inter-Domain Routing (CIDR) replaced prior classful network design, which was not scalable [69]. While the new forwarding mechanism based on CIDR (Longest Prefix Match) increased efficiency and imposed a hierarchy within addresses, the scalability concerns were renewed in the light of further developments. One such scalability issue was multihoming, which under the current architecture can cripple aggregation. Without the concept of route aggregation, the routing table size inflates.

If a single node has non-contiguous prefixes or multiple geographic locations exist within the same prefix there exists no opportunity for aggregation. Thus, single prefix spread across multiple locations or contiguous prefixes across multiple locations or discontiguous prefixes in the same location causes inflation of routing table size and increased routing table churn.

With IPv6, some of these problems are mitigated to an extend with deeper hierarchy and policies for network architecture flexibility, support for route aggregation, easier renumbering and multihoming. Out of the 128-bit address space, top 48-bits are assigned as global routing prefix, subsequent 16-bit identifies the subnet (enabling aggregation within an AS) and the last 64-bit represents the interface ID (48-bit Ethernet + 16 more bits). This approach still does not address all scalability concerns. Other proposal to support multihoming has been proposed. SHIM6 [55] is a layer 3 shim for providing locator agility below the transport protocols, so that multi-homing can be provided for IPv6 with fail-over and load-sharing properties.

Another proposal is to use various proposed namespaces simultaneously between domains which uses or shares different naming/addressing formats [70], but this approach requires further translations. This is necessary if the domains that communicate do not share a common name-space. Gateways (identical to NATs) between domains or even Specialized Services within domains can translate the identifiers used at a particular context. If flat name spaces are used, then routing becomes challenging, since the routing information of nodes with flat names is inherently difficult to aggregate.

The Ambient Networks project proposes two alternatives to approach this challenge. In the first main alternative, the top level structure of the global network consists of arbitrarily connected network domains, much like the InternetŠs Autonomous System (AS) structure. The difference is that the network domains can use different internal addressing schemes. The top-level routing problem is thus similar to the one that BGP solves in the Internet, but with the difference that address prefixes are not used, just the equivalence of AS numbers and paths. Finally, the name resolution system needs to consult top-level routing information to be able to resolve names into addresses.

Flat naming has been proposed by other projects as well like HIP [32], but scalability and efficient with such approaches is still far from ideal. One study which looks into routing over flat labels [71] reaches similar conclusion.

2.1.6 Realtime

Real time communication in a packet based based network using self-contained packets in the absence of a separate control channel is hard to achieve. The Internet architecture is also built around the concept of best effort delivery, with transport layer protocols assuring end to end delivery using resending, without much QoS support. In IPv4, QoS is achieved using the ‘Type of Service’ field or the ‘Differentiated Services Code Point’ field in the packet header. This approach classifies the packet into what kind of service is expected by the packet, while being delivered through routers across the network. The header information is used by devices in the network, to assign resources based on their configurations. However, this also means that not all QoS-compliant devices are compatible with one another.

In IPv6, provisioning for QoS is improved with header containing a new field, called Flow Label field that defines how particular packets are identified and handled by the routers. The 'Flow Label' field identifies packets that belong to a particular flow[1] and allows a router to handle them efficiently. This label ensures more efficient delivery of information from one end to another without the need for extra processing in intermediate nodes, which is helpful for peer-to-peer applications like VoIP and other real-time applications. If the architecture is viewed as a Distributed Real-time Embedded (DRE) middleware, QoS control has been proposed [72].

The emergence of broadband wireless mobile cellular and ad-hoc networks, together with increased computational power has provided the impetus for a new breed of real-time multimedia applications such as video conferences, Voice over IP (VoIP) and other multimedia streaming services to become prevalent forcing network infrastructure providers to accommodate these new paradigms. The QoS guarantees for such applications cannot be provided just by over-provisioning bandwidth alone (as was the approach till the usage caught up with and exceeded the bandwidth capabilities of the network providers). Network providers have continued to experiment with various options including preferential treatment based on pricing (tiered pricing and service), bandwidth limiting, traffic shaping. Since, QoS is an end to end characteristic for a flow (a generic term used to identify traffic streams in a network), it is impossible to provision a reliable QoS without providing it through out the length of the path, which might cross administrative boundaries. For such a QoS framework, there are various components which needs to be addressed such as flow specification, routing, resource reservation, routing and admission control. There are various protocols proposed to address these components, like Resource reservation protocol (RSVP) specifically covering resource reservation component [73]. RSVP is a receiver based resource reservation signalling protocol designed

[1]The connection session that start from a particular host and head to a particular destination

specifically to work over IP [74]. Applications can use RSVP to communicate their requirements to the network in an robust and efficient way.

2.1.7 Layerless Architecture

With the restrictions of Layered architectures in perspective, alternative proposals are being increasingly discussed within the networking community [75, 76]. The nested levels of abstraction called protocol layers is not a sufficiently flexible abstraction for network software modularity and alternatives have been proposed, such as Role-Based Architecture [38] and Haggle [36].

Haggle approaches these challenges using the concept of Pocket Switched Networks (PSN)[77, 78], while Ambient Networks [35] adopts a more complex merging of network domains (meeting of network domain). The Haggle network architecture is aimed at providing seamless network connectivity and application functionality in mobile environments by separating application logic from underlying network architecture. It is constructed around the concept of PSN to take advantage of both infrastructure based and Ad Hoc (peer to peer) communications opportunistically. To insulate the applications from changes in the underlying architecture, Haggle uses late binding to map the application parameters to underlying network architecture. Haggle suggest a new set of mobile networking principles such as forwarding using application layer information, asynchronous operation to accommodate opportunistic or intermittent connectivity, intelligent and involved middle nodes etc to accommodate the new architecture.

Data-Oriented Network Architecture (DONA) [79], borrowing heavily from other exercises like TRIAD, SFS and HIP, suggests a clean-slate redesign of Internet naming and name resolution, to address specific features such as persistence, availability, and authentication for service access or data retrieval. DONA justifies this proposal pointing out the existing discordance between historical design (host-oriented) and current usage (data-oriented). However, the clean slate approach is mostly limited to the how Internet names are structured and resolved. As with HIP, DONA replaces DNS names with flat, self-certifying names, and replacing DNS name resolution with a name-based anycast primitive that lives above the IP layer. DONA proposes that names handle persistence and authenticity, while name resolution handles availability. There are other attempts to approach the problem from enterprise perspective through Application Oriented Network Architecture (AON) [80], borrowing ideas from NGN architecture [81].

2.2 New Service Driven Architectures

The terms 'Middleware' and 'Distributed Systems' have been liberally used in various contexts [82]. Some authors use the former two as at least partial synonyms. Others point out clear distinctions between the two ideas. For Dollimore *et al* [83], middleware is 'a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming language'. Examples offered include CORBA [84] and JavaRMI [85]. Weerawarana [86] describes CORBA along with J2EE and COM as 'distributed system technologies'. For these authors middleware includes higher-level frameworks, supporting functions such as transactions, security, management, and messaging.

Erl [87] broadly includes as middleware software, systems and products that enable inter-application communication, especially in heterogeneous environments. He specifically considers EAI [88] as one type of middleware, but his definition appears to allow Web services into this role as well. Most authors agree that middleware stands between the application layer and the network layer. Less settled questions on middleware center mostly on how much functionality resides there versus at higher application layers such as those involving Web services.

The traditional distributed systems have some shortcoming like being tightly coupled object systems, centered around object technology. A Service within such systems is implemented as a methods of a class implemented by the object. Another shortcoming is the lack of interoperability. Different distributed systems are incompatible (CORBA, J2EE, COM, etc), with even the underlying framework being incompatible (transactions, security, management, messaging etc).

One method to improve inter-operability within Distributed Systems is to implement a standard based interface and a common messaging platform to communicate among them. Lack of interoperability among vertical silos or chimney design create independent islands of implementations with new requirement to integrate these islands (EAI - enterprise application integrations). We observe a similar situation in the network world with proposed future network architectures. EAI uses message oriented middleware to bridge different domains which will be a good principle to apply to network architectures.

Based on the central concept around which the architecture evolves and logical organization of software components, they can be classified into:

Layered Uses layers to simplify the architecture of a complex system. Examples include OSI and TCP/IP architectures. In a strictly layered architecture, requests flow down the layers while replies flow up the hierarchy.

Object Based Where objects (encapsulation of functionality) for the basis of the architecture with uniform procedure interfaces. [89]

Data Centered Where data plays the main role. Examples include DONA [79] and Haggle.

Event Based Revolves around the occurrence of events [90] such as Publish/Subscribe systems [91].

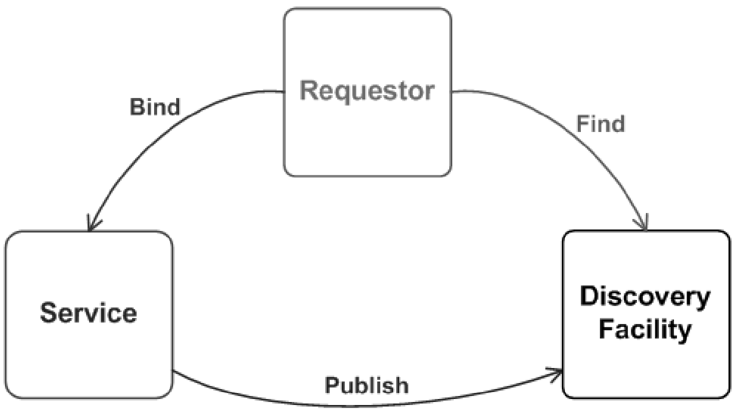
Service Oriented Where the main paradigm is the Service. Services are published, discovered, composed and used depending on the context within the architecture.

2.2.1 Service Oriented Architecture (SOA)

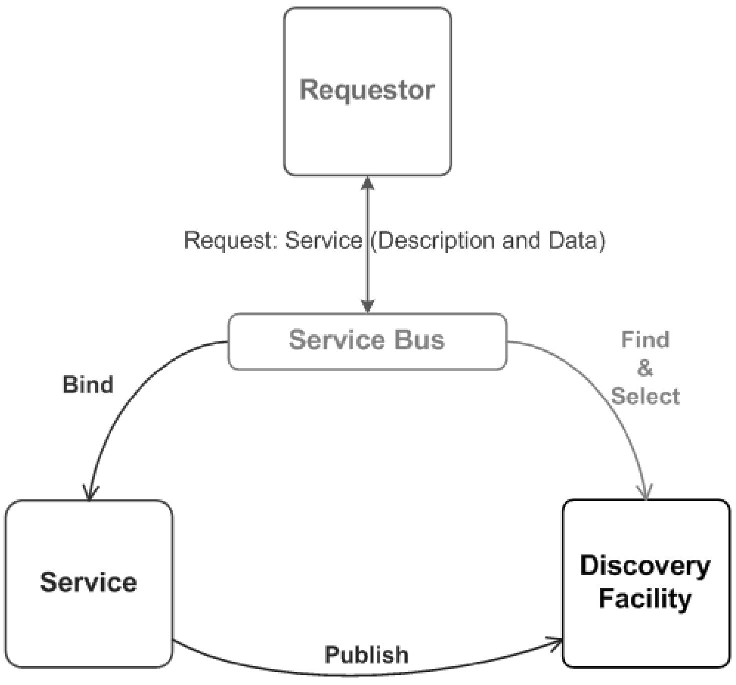
SOA represents an abstract architectural concept of building software systems that are based on loosely coupled components (Services) that have been described in a uniform way and can be discovered and composed [86]. Loose coupling precludes any knowledge or assumptions regarding the specific platform, implementation, formats or protocols. The services that form the part of an SOA should be dynamically composable by any entity interested in availing it. The core elements that comprise an SOA is illustrated in Fig. 2.4(a).

The figure represents the basic building blocks in an SOA approach. The concept revolves around Services, which can be user by an entity called *Service User*. The Services are made available by the *Service Provider* and is published in a *Service Registry* to be discovered by potential Service Users. A service registry organizes information about Services and provide facilities to publish and discover services. Services can be reused by various Users and the service registry facilitates reuse. It refers to a place in which service providers can impart information about their offered services and potential clients can search for services.

The SOA architecture can further extended with a Service Bus (SB) (Fig. 2.4(b)) to make the discovery and binding process more transparent to the requesting service by visualizing the candidate services from the requestor's perspective. The Service Bus virtualizes the candidate services from the requestor's perspective. The Service Bus accepts the service description from the requestor, uses discovery facility to find out relevant candidate services, selects one of them, retrieves the necessary binding information, binds the service accordingly, transforms the input data from the requestor accordingly, send the request to the message to the service, receives the response which is passed on to the requestor. The concept of SB is important as it forms a decision making entity or a middleware on behalf of the requesting service. The SB "*accepts the service description from the requestor, uses discovery facility to find the list of qualifying services, selects one out of them, retrieves necessary binding information, binds the service accordingly,* *transforms the data from the requestor properly, send the corresponding request message to the service, receives the response and passes this response in the proper format to the requestor*" [86].



(a) SOA Triangle.



(b) SOA Triangle enhanced with Service Bus.

FIGURE 2.4: Basic components of SOA architecture

Despite the prevalence and adaptation of SOA in enterprises over the years, the full potential of SOA (like dynamism, adaptivity) is still not fully exploited [92].

Future Proofing

The concept of wrapper/interface isolates the service implementation without impacting the other parts of the system. This is a very desirable property. Allowing this property to be applied dynamically provides flexibility to configure networks and components dynamically. This feature requires that there exists a universally agreed standard to approach to describe (in a machine readable way) the wrappers or interfaces that the services provide. The Web Services Description Language (WSDL), within the context of Web-Services provides this capability. While Web-Services are not the only approach to realize SOA, it is certainly the most accepted and widely a adopted by industry[93].

There are different methods to realize SOA in communication systems. Web-Services represent one important approach and is the most adopted and widespread within the IT industry. There are various other Middleware (OMG CORBA, MSDCOM etc.) that can be used for such abstraction, but Web-Services have marked advantages like being much more loosely coupled, dynamic and adaptable to change. Besides, it supports and open way to develop specifications and using technology via a broad consortia, which takes into account the stakeholder interests[1].

Web-Services

We note the best known implementation of SOA in Web Services (WS-*) [94] specifications, standardization and its implementations. While, Web Services specifications specifically address Enterprise Application Integration (EAI [95]), we need a similar principle and framework to be applied to future heterogenous communication networks in order to interconnect business borders and administrative domains.

The world wide consortium (W3C) defines Web-Services as:

[1]While the specifications are still susceptible to evolution and further standardization, the design supporting each of the specifications are unlikely to change in fundamental ways

> *A Web-Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using Simple Object Access Protocol (SOAP) messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

Turning to the definition of Web services, [83] describes these simply as ‘a collection of operations that can be used over the Internet’. Weerawarana [86] tightens this focus to mean, more precisely ‘a standards-based, XML-centric realization of an SOA’ (Fig. 2.5). Web services may range anywhere from minimal XML messaging over HTTP [87] to a comprehensive SOA. All Web services definitions agree that Web services reside at the application layer and involve some sort of messaging over Internet protocols.

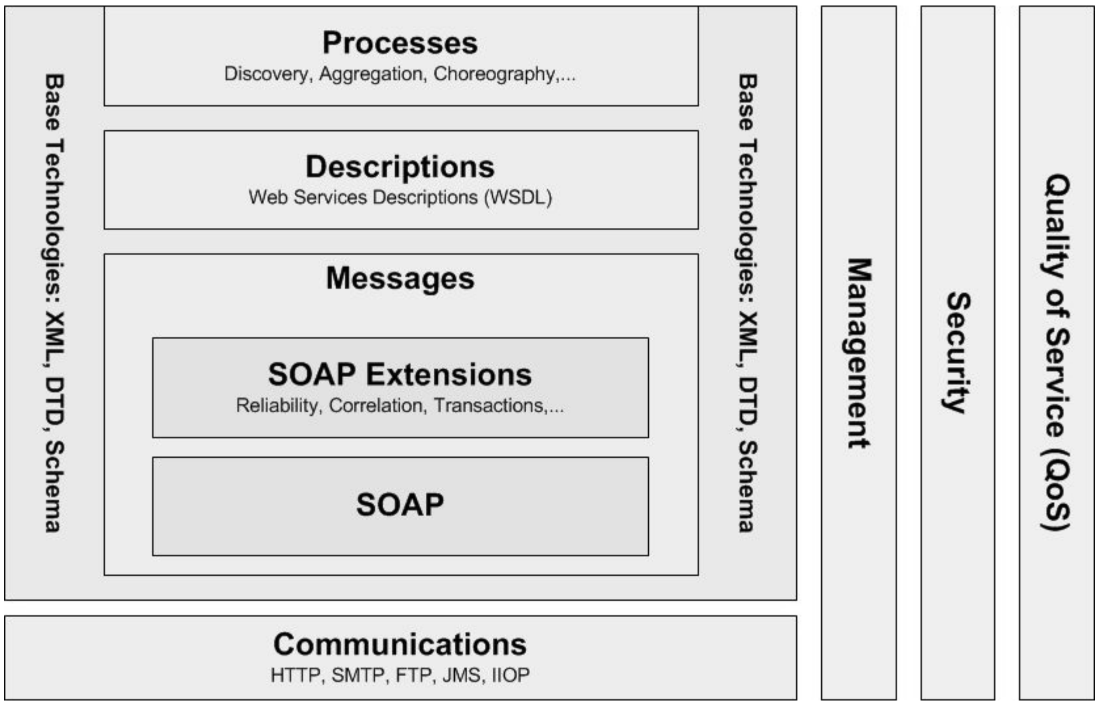

Processes
Discovery, Aggregation, Choreography,...
Descriptions
Web Services Descriptions (WSDL)
Messages
SOAP Extensions
Reliability, Correlation, Transactions,...
SOAP
Base Technologies: XML, DTD, Schema
Base Technologies: XML, DTD, Schema
Communications
HTTP, SMTP, FTP, JMS, IIOP
Management
Security
Quality of Service (QoS)

FIGURE 2.5: Web-Services stack

Web-Services provide a uniform way of describing components or services with in a network, locating them and accessing them. Web-Service specifications define formats and protocols that allow services to interoperate across those vendor platforms that provide conforming implementations, either natively or by mapping them onto existing proprietary middleware offerings. The standards and specifications that are adopted are being developed in an open way through community organizations like W3C and OASIS. The process allows for a consensus based standardization and vetting by commercial interests before being accepted or approved as a standard. Web services rely on XML for the basic underlying data model, SOAP for the message processing and data model, and WS-Addressing for addressing services and identifying messages independent of transport. Web-Services are inherently transport neutral, which means one can use any communication protocols, including proprietary ones or the widely used HTTP/HTTPS.

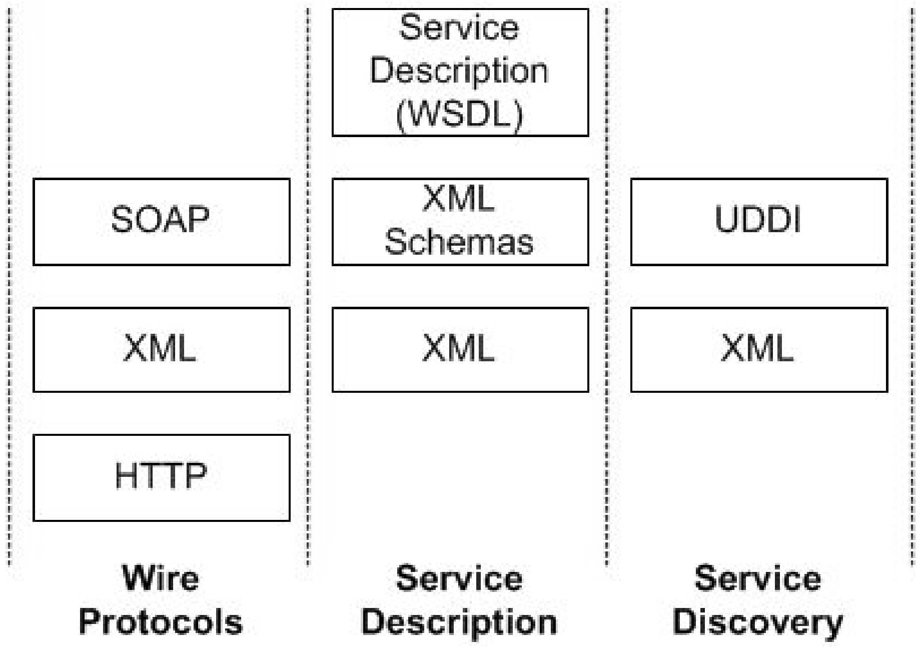


FIGURE 2.6: Web-Services protocols based on XML

A Web-Service can be associated with temporary roles, depending upon its utilization during runtime. It could act as a service provider when it receives and responds to a service request. It can in turn act as a service consumer when it need to send requests to another Web-Service. Or, it could take up both roles during a service composition. Web-service framework provides a communication framework based on physically decoupled contracts allowing each service contract to be fully standardized independent of its implementations providing a potentially high level of service abstraction.

Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP)[96] is a simple and extensible XML based communication protocol that provides a way to exchange structured and typed information between applications running on different operating systems, with different technologies and programming languages. It defines and extensible enveloping mechanism containing three distinct elements: *an Envelope a Header* and *a Body* [96]. With SOAP, the underlying transport might change as the message is routed between nodes, even the mechanism selected for each hop may vary as required. An important facility is the feature that the messages can be routed based on the content of the headers and the data inside the message body. SOAP forms the messaging framework of ROSA, owing to these attributes. Another specification, WS-Addressing provides an interoperable transport independent way for identifying message senders and receivers associated with a message exchange. Besides securing end-to-end endpoint identification in messages, this specification enables messaging systems to support message transmission the networks that include middleboxes like endpoint managers, firewalls, gateways etc. Further details of these specifications are available at [94, 97].

The middleboxes mentioned above can be fitted into the category of SOAP *Intermediaries*. Intermediaries are entities positioned between a client and service provider that provide additional functionality and value-added services. For example, Intermediaries can offer functionality like customization, personalization, caching, filtering, and transcoding by modifying and enhancing data as it flows between a Web-Service User and the Provider. Intermediaries intercepting messages, perform their designated functions and subsequently forward the updated message towards the ultimate receiver. SOAP messages can be routed through numerous intermediaries on it way to the destination node.

Intermediaries provides a way to extend the functionality of the Service Provider and User. They also offer flexibility, since they can be dynamically added and removed. The

SOAP messages can be routed based on the content of the header as well as the message body making enabling highly flexible routing. Web services protocols (in particular SOAP) provide clean, decentralized, and modular support for intermediaries. Intermediaries themselves can be implemented as SOAP services. Extensibility has always been one of SOAP's major design goals, and intermediaries are supported by using the SOAP extensibility model.

Web Services Description Language (WSDL)

WSDL [98] provides a model and XML format for describing what a Web-Service offers. This standard Service description in WSDL isolates the Service functionality from the implementation details of the Service. WSDL models Web-Services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information [99]. WSDL describes a Web-Service as an abstract and a concrete definition. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (Services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate [99]. The metadata provides the abstract definition of the information necessary to deploy and interact with a Web-Service.

Universal Description Discovery and Integration (UDDI)

UDDI [100] provides an infrastructure that supports the description, publication, and discovery of service providers; the services that they offer; and the technical details for accessing those services. A core aspect of UDDI is how it organizes information about services and the providers of services. Information entities (UDDI data) are organized in a data model and stored in a UDDI service registry. Inquiring (search and lookup entries) and publication (publish, delete, and update registryŰrelated information) are core APIs. There are three common implementations of UDDI based on accessability: Public, Intra-enterprise (within a domain) and inter-enterprise (among domains).

The Service registry is enhanced with Service categorization to distinguish services and classify them according to one or more categories. Services whose description includes similar or related concepts, but different syntax, can cause confusion during the discovery phase by Service users. The semantic research community has proposed alternatives to enrich service descriptions semantically and enhance classification schemas in services registries. Basic taxonomies can be enriched or replaced by ontologies. Ontologies structure concepts within a domain and define their meaning. Axioms constrain possible interpretations of concepts and reasoning mechanisms that support inferences from existing data. However, several issues still must be addressed, such as reducing the computational cost of reasoning, maintaining the ontologies, and refining search results to improve effectiveness [101].

Web-Service Extensions (WS-*)

Apart from the above mentioned specifications, the Service descriptions are enhanced by various extensions. For example, policies can indicate the constraints, permissions and other requirements associated with a specified Service. A Policy service can use the WS-Policy [102], an extensible framework for Web-Services constraints and conditions allowing for a clear and and uniform expression of of the available options.

Another WS specification, *WS-MetaDataExchange* extension supports the dynamic exchange of relevant metadata for service interaction directly between interacting Web-Service endpoints without the need for third-party registries or discovery facilities. The extension *WS-Reliability Messaging* enables message reliability checking, as the name indicates. *WS-Addressing* [59], through extensible XML elements, provides an interoperable, transport independent way to identify message senders and receivers associated with a message exchange. It decouples the address information from specific transport used by providing a mechanism to place the important address information directly within the Web-Service message. This specification enables messaging systems to support message transmission through networks that include middleboxes like end-point managers, firewalls, gateways etc over heterogenous transport mechanisms.

WSDL by itself does not support rich expression of QoS associated with a Service. But, it is possible to express parameters through extensions such as Security (WS-Security) [103], transactions support (WS-Transactions)[104][1], message reliability (WS-ReliableMessaging)[105] etc. The Web-Services recommendations and standards facilitate applications to utilize various interaction patterns like Asynchronous send/Receive and Anycast apart from traditional Synchronous request/response model. This support can be built into the middleware in a uniform way. Besides, the delivery based on SOAP enables applications to have fine tuned parameters on message delivery that allows SOAP messages to be reliably delivered between distributed applications in the presence of software component, system, or network failures (WS-ReliableMessaging). Such extensions allows applications to make use of delivery options such as AtLeastOnce, AtMostOnce, ExactlyOnce, InOrder without handling the related messaging themselves.

The process of delivering SOAP messages through a series of nodes (called SOAP Intermediaries) is called SOAP routing and is defined in the WS-Routing protocol. In SOAP routing, each intermediary provides value added functionality such as logging, validation, auditing etc apart from the basic provision of a message delivery channel. SOAP Routing Protocol (SOAP-RP) is a stateless protocol for the exchange of SOAP messages supporting various patterns (one-way, request-response, guaranteed-response, peer-to-peer) independent of the underlying protocols.

Interoperability requires a paradigm, programming language and platform agnostic messaging communication channel/layer/middleware. The internet is an example for such an abstraction using TCP/IP, DNS, HTTP etc. Web-Services extends this capability using XML, SOAP, WSDL and UDDI [106].

[1]WS-Transaction or WS-TX consists of three separate specifications[104]: WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity

2.3 Discussion

The goal of middleware technologies discussed in section 2.2 was interoperability. Web-Services, based on SOA, share this goal, but aim to achieve interoperation in a simpler, more standards-based manner. But, there are disagreements regarding the positioning of Web-Services with regards to traditional Object Oriented Middleware. In certain cases (reliability checking, for example) a certain overlap of functionality may occur when Web-Services begin to take on roles formerly played by middleware. This overlapping may occur in functionalities fulfilled by traditional middleware like compression, encryption, management, transactions, and other critical functions of an end-to-end solution. Web-Services can even provide functionality formerly provided by middleware or even by lower levels of the network stack.

Stal [107] argues that even though Web-Services deliver on the integration of heterogeneous islands of information, they lack certain aspects of Object Oriented Middleware and hence is not a replacement, but a complement to the latter[1]. This shortcoming of Web-Services necessities the role of object-oriented middleware to integrate back-end solutions in an SOA. Since the strong point of Web-Services over traditional middleware is interoperability, it is vital to avoid incompatibilities within its specifications, standards and its implementation.

WS-Interoperability (WS-I) has published a WS-I Basic Profile that defines a set of cross-platform standards to promote and ensure interoperability. During the time of writing,Web-Services Interoperability Organization is working on the working draft for Basic profile version 2.0 [108]. [2] Sheth and Miller [110] argues that Web-services has the potential to have a revolutionary impact since it is incrementally implementable, has low barriers and costs of entry as well as a standards-based approach. They also

[1]He notes that despite including the word "Object" in its name, SOAP and other XML Web services languages lack notions of inheritance, polymorphism, and even of objects themselves.

[2]Some of the misconceptions regarding Web-Services are clarified by Vogal in '*Web Services Are Not Distributed Objects*'[109].

argue, however, that beyond the initial adoption, there exists a need for Semantic Web technologies for Web services over the long term.

2.4 Summary

After an inspection of the state of the art in emerging communication and networking paradigms, we conclude that a layer-less network architecture is an ideal option for a future generation network architecture. In order to mitigate the challenges faced in complexity when the layers are blurred, we would need a abstraction to separate functionality and provide flexibility in the architecture. Cross-layering fails in this regard as it is impossible to propose a cross layer solution which will address all the new requirements using an uniform approach, without breaking the layered paradigm (thus making the layers moot) or avoiding extreme complexity. We propose to use a Service Oriented approach as the unifying solution and Services as the basic abstraction to mitigate complexity. We still need to extend these principles and propose new tenets to accommodate a new network architecture, but having a proven and widely used paradigms form a reliable base to build up on. We will inspect the aspects of Service Oriented approaches that we adopt in the subsequent chapter.

CHAPTER 3

ASPECTS OF SERVICE ORIENTED DESIGN FOR AN ARCHITECTURE FRAMEWORK

In their paper introducing ‘Scenario-Based Analysis’ of Software Architecture, Kazman *et.al* [111] discusses the need for Architectural Analysis from a software system point of view[1]. This is an evolved version of the Software Architecture Analysis Method (SAAM) [112] to include structured scenario based analysis. SAAM was a precursor to the Architecture Tradeoff Analysis Method (ATAM). The architectural analysis forms the basic study regarding the validity of the requirements and strategy adopted for the proposed architecture.

The quality of the final system can be seen as the summation of the quality achieved at the various stages of the system development life-cycle. While it is easy to observe that the lack of quality at any of the stage can adversely affect the final system quality, it is difficult to qualitatively measure the positive contribution at any stage. There is

[1]Specifically, see section *Architectural Analysis: Critical Tool* and related references

a consensus, however, that any quality that the system needs at the end, will have to be provisioned at the design stage. That is to say that, if the quality of the system can be measured or is dependent on a set of quality attributes, then the architectural phase plays the major role by provisioning the attributes into the design. A lack of quality (in quantitative terms) at the architectural stage with regards to accommodating such attributes cannot be corrected at later stages, but by revisiting the architectural stage again. More experienced narrations are available in [113].

This discussion is important when we talk about continually evolving systems such as communication network architectures. For instance, the design constraints or quality attributes considered during the design of the IP network (around the 60s and 70s) might be superseded by more attributes and requirements at present. Similarly, future network architecture requirements might be different and unpredictable at this point in time. This is evident from the humorous admission by Tim Berners-Lee (one of the major contributors to the inception of World Wide Web) that "the double slash ('//' after the 'http:' in Web addresses), though a programming convention at the time, turned out to be unnecessary [114]. While this is not a major design flaw, it points to the fact that some of the conventions we use extensively are no more than rules of thumb or personal preferences from a few academics from the 1970s. There are significant shortcomings too with the present network architecture, as we have seen earlier.

The question we need to address is '*whether it is possible to design an architecture with appropriate or enough quality attributes at design stage (present) to accommodate inclusion of future attributes*'. This invariably points us to a 'plug and play' architecture for architectural components. To study this possibility, it is important to analyze the relationship between the quality attributes coined during the requirement stage and how that translates via architecture design to system quality.

Functionality and quality attributes are orthogonal [113]. This assertion helps us to examine the relation between quality attributes and functionality of systems. The important insight we can acquire is that attributes and functionality can evolve independent of each other. This indicates that it is possible, theoretically at least, to design a system with similar quality attributes, but varying functionality to accommodate various requirements. However, orthogonality here does not mean they do not influence each other at all. Just because an attribute has been provisioned in the design, it will not necessarily translate into any level of functionality. For example, security as a quality attribute will not necessarily lead to the end system being highly secure. Conversely, for a system to be secure it helps significantly if security is considered as a quality attribute and design stage accommodates for its provision.

3.1 System Architecture Analysis and Evaluation

The architecture development process need to follow some standard Development Life Cycle (DLC) process for modifiability and to clarify the rationality behind decisions made [1]. There are many existing architecture centric analysis and design methods that can be utilized for our architecture [115], some of which are listed below:

- Scenarios-Based Analysis of Architecture [111]
- Quality Attribute Workshop (QAW) [116]
- Cost-Benefit Analysis Method (CBAM) [117]
- Active Reviews for Intermediate Designs (ARID) [118]
- Attribute-Driven Design (ADD) Method [113]
- Architecture Tradeoff Analysis Method (ATAM) [119]

Besides being architecture centric, these methods share some common characteristics. These methods use scenarios to drive the direction and focus of the method's activities.

[1]When the life cycle development method specifically targets software architectures, the term used is *Software Development Life Cycle*

These methods all focus of how certain quality attributes influence the architecture. One of the most important features is the documentation of the rationale behind the decisions made which serves as a knowledge base on which current and future decisions are made. Besides, all these methods elicit multiple views involving various stakeholder resulting in the architecture.

While the representation might vary, a typical Systems Development Life Cycle (SDLC) contains the (al least) the following steps Fig. 3.1 [113, 115]:

1. Initiation/Planning
   (a) Understanding the business case/need and constraints
   (b) Understanding the requirements
2. Architecture Design
   (a) Creating/selecting the architecture
   (b) Detailed design of the architecture
   (c) Analysing, Evaluating and Documenting the architecture
3. Implementation
   (a) Implementing the system based on the architecture
4. Testing
   (a) Ensuring that the implementation conforms to the architecture
5. Deployment
6. Maintenance

These activities do not imply any particular development process and is generic enough to be accommodated into any development process such as waterfall, spiral, etc. In reality, systems development is a complex, continuous, iterative, and repetitive process,

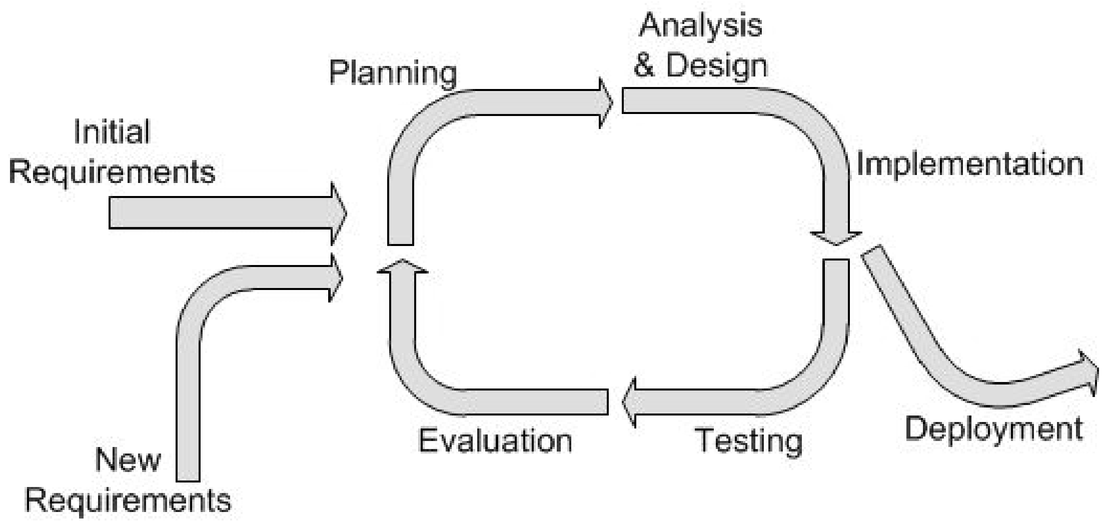


FIGURE 3.1: The Iterative and Incremental Development (IID) model

specifically in our case. The simpler development models like waterfall model do not reflect this complexity. Since, the architecture will evolve in the the future to accommodate new paradigms, we propose the Iterative and Incremental Development (IID) approach as a model for the architecture development. Such an approach is essential to *avoid a single-pass sequential, document-driven, gated-step approach* [120].

Separating an architecture development process into interim steps to specify partial architectures, provides us with milestones and checkpoints where the output can be evaluated and the next iterations substantiated. These intermediate steps are Frameworks, Reference Models, and Designs which can be constructed in a top down approach. A typical example of this process is where a Framework is used to derive a Reference Model, which is then used in a particular Design which in turn results in an Artefact, such as a piece of delivered software Fig. 3.2. There exists no one-to-one relationship among these, as multiple Reference Models can be derived from a single Framework and a single Reference Model can be drawn from multiple Frameworks.

The core task of creating a Framework is to define a broad set of Services,i.e, a Framework consists of a set of Services defined at various levels of detail. The process of partitioning the required functionality into service boundaries (defining Services) is termed '*factoring services*'[5]. The granularity of the defined Services can vary depending on

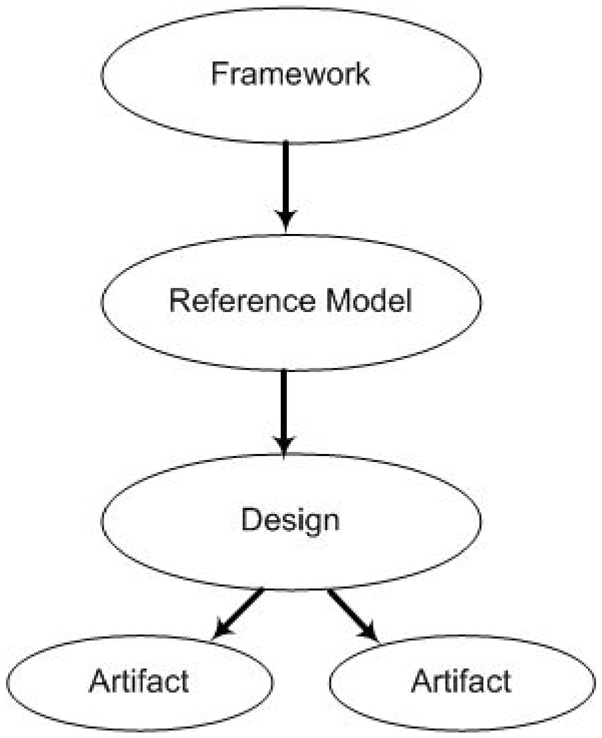


FIGURE 3.2: Relationship between Models

the attributes and factors considered for its design. As a rule of thumb, more fine-grained Services provide more flexibility for Reference Models and Designs to group Services for a specific purpose. As the design considerations and factors influencing the factoring is likely to change over time, the Factoring of Services themselves is an ongoing process. This is a common theme in SOA, since experience informs the choice of Services, identifies gaps, and indicates that Services require splitting or joining [121]. There is no current 'best practice' for factoring Services within a Framework, but one rule of thumb is to define Services that are fine-grained in overall scope, yet whose operations are still capable of encapsulation for efficient remote access.[1]

[1]There is no current best practice for factoring Services within a Framework, but one rule of thumb is to define Services that are fine grained in overall scope and define a single behavior or step within a composable architectural process[121].

3.2 Towards a Service Oriented Approach

Issues with current distributed systems models prevent them from being usable for a next generation network architecture. Tight coupling and lack of interoperability among various existing middleware causes islands of incompatible implementations. We develop our vision from a top down approach, from the point of view of the application developers. From such a perspective, networking is not just connectivity specified by an end point tuple (as in Berkeley APIs). A network is a collection of distributed *services* that are *available* to the applications. The application developer should not worry about the state of the network at application runtime during the application design. This decoupling of addressing network end points directly in applications can be achieved through a uniform approach to exploit the underlying network infrastructure via a generic, rich and standardized interface. Such an interface provides abstraction of network capabilities to applications and decouples the heterogeneity arising from the below mentioned factors. Adapting to heterogeneity forms one of the basic characteristics of our approach. Heterogeneity in various layers of the current architecture arise from various sources including:

- Network technologies, devices and Operating Systems
- Middleware solutions and communication paradigms
- Programming models and languages
- Services and interface technologies
- Domains and architectures
- Data and document formats
- Nonfunctional aspects such as information models, security, availability, transactions etc
- Business borders

- Communication procedures and security policies

This high level abstraction for a network has various advantages. Such an approach inherently accommodates the business boundaries existing in the real world networks such as commercial boundaries, administrative domains etc and helps traversing across these a natural part of service negotiation and usage. Usage of resources such as processing power, storage, bandwidth etc can be considered as services thus can be described via web services. This can be considered as service oriented middleware.

3.2.1 Network as a Service

We state that the layered paradigm in networking architecture is not the ideal path for abstraction, as discussed in chapter 1. As with 'Haggle', we propose a layer-less architecture which abstracts the underlying connectivity and network computational resources to applications via a high level API. Applications should not be burdened with attaching themselves directly with end points or the connectivity status of various available interfaces; it should be automatically taken care of by the underlying architecture. We argue that network connectivity is a service for any application to use which is the orchestrated from underlying (possibly orthogonal) capabilities of the node and the environment that the application presently resides in. This enables us to look at architecture from a service oriented point to view. This high level abstraction for a network has various advantages. Such an approach inherently accommodates the business boundaries existing in the real world networks such as commercial boundaries, administrative domains etc and helps traversing across these a natural part of service negotiation and usage.

3.2.2 Network Service as a Collection of Services

The next step is to identify the basis with which we compartmentalize the capabilities into modules, which can be later orchestrated. We take cues from the 'tussle' [7] being

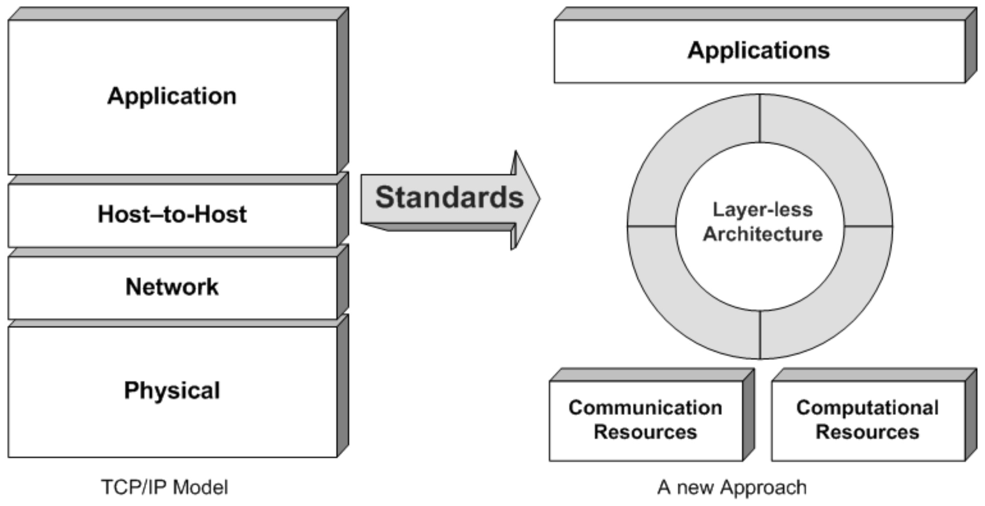


FIGURE 3.3: The vision of *Network as a Service*.

played out in the networking world to impose boundaries on the modules. A module encompasses a well defined service small enough to be a factor in the tussle but large enough to provide a specific, well defined, usable and non-trivial service. Identifying a set of modules which provides the services needed to compose a network architecture is not straight forward considering the fact that as a 'future' network architecture, it should be able to gracefully accommodate a wide spectrum of potential uses and abuses, both those encountered in the present scenarios as well as those possible in the future. The borders of modules and which attributes to consider as a module is of course, an open question. Different approaches can propose different modules to accomplish the same goals and it might be probable that it is impossible to agree on a standard set. But, as long as different modules provide a uniform interface and a clear description of its capabilities, the architectural principles we propose hold true.

From the above research directions, a sense of future direction can be derived. We can see not one 'Internet' but many 'Internets' of varying capabilities and characteristics (inter-network of things, inter-network of specific applications, inter-network of specific intentions etc.). This concept is already starting to evolve if we consider Peer-to-peer application 'networks' using the existing infrastructure as a transport network. This brings us very close to a similar problem that existed in enterprises towards the beginning of the 21st century. Enterprises built their IT systems to streamline their processes.

Large distributed enterprises built middleware to support transactions and interconnect their systems across domains. Since there were no standards for such systems, these proprietary systems posed a problem during instances of interoperability (mergers, acquisitions, collaborations etc). The concept of Service Oriented Architecture (SOA) [97] was adopted to enable a standardised and open way for enterprises to open up their IT infrastructure for collaboration. We note the best known implementation of SOA in Web Services (WS-*) [94] specifications, standardization and its implementations. While, Web Services specifications specifically address Enterprise Application Integration (EAI [95]), we need a similar principle and framework to be applied to future heterogenous communication networks in order to interconnect business borders and administrative domains.

Virtualization is a another approach to mitigate the heterogeneity. The abstraction of capabilities as services with generic interfaces helps facilitate virtualizing underlying capabilities across domain boundaries, without the need for applications to be concerned about the platforms on which they might implemented. We take clues from the concept of application layer implementations of distributed web services. The simple sounding goal of 'connecting to customers, suppliers or partners electronically' [122] translated into web services that offer standard based mechanisms to create or compose services from composite and often cross-organizational components and Web services [123]. We look at the underlying network architecture under the same requirement considerations, i.e, a service to be offered to applications, composed of various other services, local or distributed.

3.2.3 The Principles of Service Orientation

Before we apply service orientation principles to the network architecture, it is helpful to identify the definition of a *Service*. A Service is defined as 'an entity with a specific *Functionality* which can be *Described* using metadata and can be *Discovered* by Users who can *Compose* them to use the fucntionality[86]'.

If a domain abstracts its services/infrastructure so that it presents its functionality clearly in the form or loosely coupled coarse-grained services, then other entities can use the services independent of the underlying platform. Moreover, this approach helps an IT infrastructure to meet yet unknown requirements which is not easily attainable using traditional IT planning and design methodologies.

One pattern that enterprises use to facilitate SOA is via the use of Enterprise Service Bus (ESB)[124]. We adopt the *concept* of ESB (Fig. 2.4(b)) and not the notion of proprietary middleware (or proprietary definitions) that major solution providers propose under the collective term ESB. From an application developer's point of view, the ESB has the intelligence to manage most of the common network related tasks, but we maintain that this should be done in a standardized and open approach.

The next step is to identify the basis with which we compartmentalize the capabilities into modules, which can then later be orchestrated. We take cues from the 'tussle' [7] being played out in the networking world to impose boundaries on the modules. A module encompasses a well defined service small enough to be a factor in the tussle but large enough to provide a specific, well defined, usable and non-trivial service. Identifying a set of modules which provides the services needed to compose a network architecture is not straight forward considering the fact that as a 'future' network architecture, it should be able to gracefully accommodate a wide spectrum of potential uses and abuses, both those encountered in the present scenarios as well as those possible in the future.

The boundaries and definition of modules can be defined based on the study of the current as well as the emerging views on networking architectures. For example, TCP/IP provides clues to successful abstractions, albeit through layering. HIP and similar proposals argue for a separation of 'names' and 'locations'. DONA and TRIAD argues for flat labels, which has interesting implications arising from ideas such as routing on flat labels [71] and mobile ad-hoc networks where a hierarchical naming convention is not required and short labels have obvious advantages. Thus naming and addressing and subsequent routing form separate tussle spaces and hence different modules.

In this section, we will focus on the argument that the given modules can be composed into a coherent architecture via a single paradigm, namely *Relationships*. We define 'relation' as an association among dynamically collaborating nodes, devices and services in a network. A relationship description contains parameters ('relationship metrics') to express the nature and background of the collaboration. For example, the intention of an application with the network can be translated as a relationship between the underlying architectural modules and provided as a service to the invoking application. This not to be confused with the relationship attribute in the Haggle architecture which is a class of meta-data used to tag a haggle Data Object (DO)[77].

While the modules represent different services abstracted for applications, the architecture itself needs to present a coherent set of high level APIs that the services can utilize to take advantage of the functionalities. We take clues from the concept of application layer implementations of distributed web services. The simple sounding goal of 'connecting to customers, suppliers or partners electronically' [122] translated into web services that offer standard based mechanisms to create or compose services from composite and often cross-organizational components and Web services [123]. We look at the underlying network architecture under the same requirement considerations, i.e, a service to be offered to applications, composed of various other services, local or distributed.

3.3 Service Terminologies

A '*Service*' is a unit of logic [5] or a publicized package of functionality [86]. Services are characterized by certain attributes, which define how the functionality is made available for use by other entities:

Functionality : Services implement some functionality.

Composable : Services are composable, which implies that an entity can use the service depending on the conditions specified either directly or as a part of another service.[1].

Describable : Services can be fully described using their metadata or other methods.

Discoverable : Services are discoverable based on their descriptions, policies for use etc.

We use 'Service' in this thesis within the context of service orientation and service oriented solution logic. The relation between Services, Service Orientation and Service Oriented Solution Logic is indicated by Erl[5] as:

> "*Service-orientation* is a design paradigm comprised of a specific set of design principles. The application of these principles to the design of solution logic results in *service-oriented solution logic*. The most fundamental unit of service-oriented solution logic is the *service*."

A service is a pattern that can be used to solve a specific problem and can be defined with in a framework at different levels [125]. Some examples of entities that qualifies as a Service is listed below. According to [126], a Service can be (among others):

- A single application with a well defined API (which also includes wrapped applications from legacy codes). In this case, the software application is offering some service (which may vary in granularity from a simulation kernel to a molecular dynamics application, for instance). In order to run such an application, it may be necessary to configure the environment, and make available third party numeric libraries on the host where execution is requested. Furthermore, execution of the application may also involve the need to have dynamic link libraries available on the host platform, and for these libraries to be compatible with the compiler used to

[1]This characteristic is clarified in later sections.

create the application executable. Seen as a service, these dependencies are hidden from the user needing access to the service and the underlying environment which supports the service is required to handle these dependencies.

- A single application used to access services on other resources controlled by a different administrator
- A collection of coupled applications, with well defined dependencies
- A software library, with a number of sub-services, which are all related in some functional sense. For instance, a graphics or a numerics library etc
- An interface to a third party software library
- A software library for managing access to a resource (this may include access rights and security, scheduling priorities, license checking software etc)

All services are not of similar scope or functionality. They can be categorized based on their functionality (logic they encapsulate), the reusability of the service and the relation of the function (dependencies on other Services) within the domain. These classifications or models help to scope the services into logical abstraction layers [5] :

Utility Services : forms the basic level of services. These services implement highly reusable and cross domain functionalities which are ideally application agnostic. Examples of such services are logging, alerts etc.

Entity Services : represent domain centric services, ideally composed of various Utility Services and encapsulates a more domain specific functionality. These services are reusable across domains using similar logic and functionality, to a large extend. In enterprises, these services represent business centric services [5].

Task Services : indicate high level domain or business tasks, which are specifically associated with the parent process or business task respectively. These services have less reuse potential, since they implement highly domain specific tasks and are usually responsible for controlling and composing various agnostic entity/utility services.

These services form three distinct logical layers as seen in Fig. 3.4. The service models (utility, entity and task) are generic enough to be applied to any type of domains.

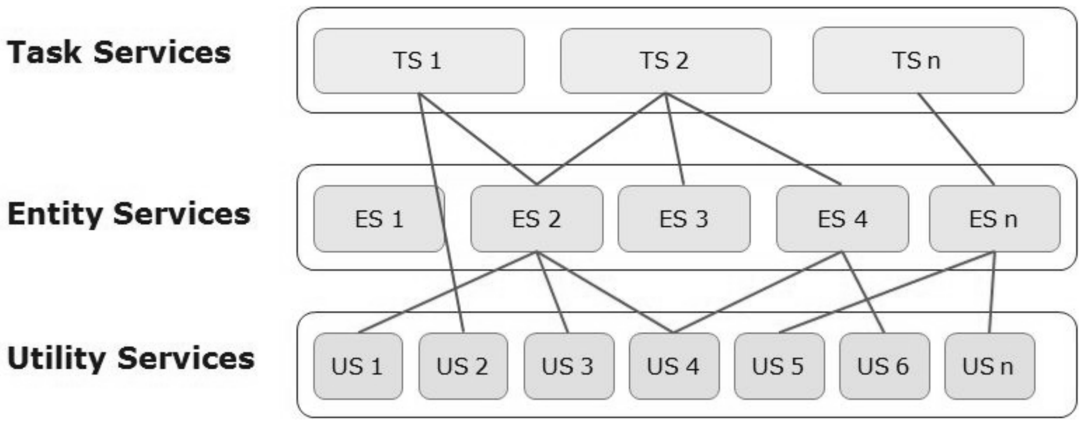


FIGURE 3.4: Service Models based on scope

Common to all service models is the concept of Service Characteristics, which primarily consists of the functional aspects (what it is and does) of the service (using WSDL in Web-Services). Apart from this, constraints and conditions regarding its usage are specified via policies. This information should be made available for other services to use. It is useful to extend this basic information with *Deployment Descriptors*[1] [5] to indicate additional information necessary to deploy the services in a working solution.

We use the service modeling process to create candidate services[5]. Our approach followed the Service factoring and identification based on common use cases[127]. See how the Services are identified based on use cases[2] in Fig. 3.5. These candidate services are documented in the service inventory blueprint. These services are further defined and their descriptions and capabilities included in the service inventory. Implementations are then connected to these descriptions.

[1]Examples of deployment descriptors may include QoS parameters, other parameters which determine the behavior of the component with respect to framework specific transactions like state management, dependencies on other services, parameters governing application specific contexts etc

[2]This is an intermediary step in domain analysis before formalizing Service Definitions

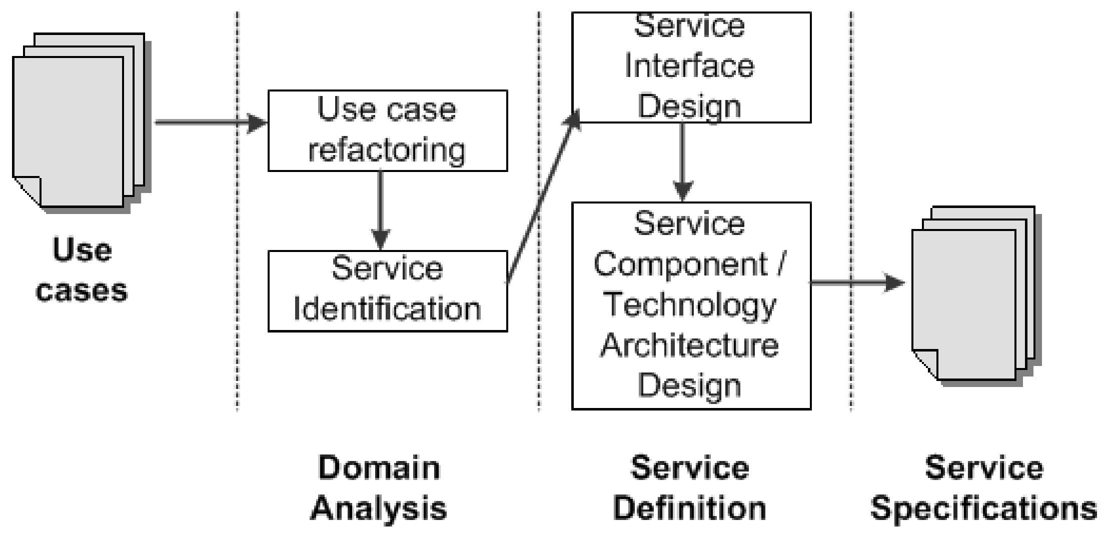


FIGURE 3.5: Service Modeling process based on use case models.

3.3.1 Defining Services

Within the context of a Framework, Service refers to a pattern that can be used to solve a particular type of problem. A Service may be realized in a number of ways, including as a Web Service, but also as an API, or as a manual work-flow.

An important characteristic of this Service Oriented approach is that its is agnostic to the implementation technologies and the processes it forms part of. The significant factor here is the collection of defined interactions that a Service supports, i.e, the functionality and expected behaviour. Services can be characterized within a Framework at several degrees of resolution[121]:

- As a definition of function and scope (in descriptive natural language)
- As an abstract model of data and behaviour (for example, in UML)
- As a data representation specification (for example, using XML)
- As an Application Programming Interface (API) specification (for example, in Java)
- As a Web Service specification (for example, using WSDL)

The set of definitions taken together can provide enough details to create a fully featured Web-Service implementation, although it is not necessary. The granularity of the definition in the list above is progressive with each level adding more details specification towards realization of the Service Fig. 3.6.

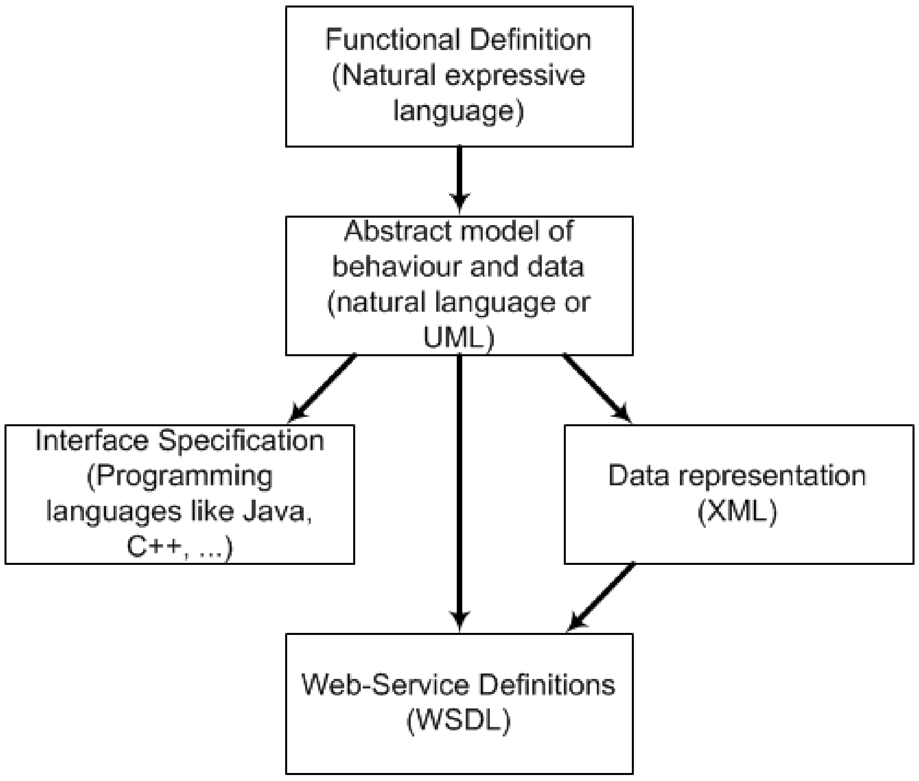

Functional Definition (Natural expressive language)
Abstract model of behaviour and data (natural language or UML)
Interface Specification (Programming languages like Java, C++, ...)
Data representation (XML)
Web-Service Definitions (WSDL)

FIGURE 3.6: Defining Services at various levels

The functional definition of the Service is expressed in descriptive natural language and forms the basis for the all other definitions. An abstract model that lays out a more formal definitions of the expected behaviors of the Service Instance together with the information structure it deals with can be derived from this functional definitions and expressed in a more formal language such as UML. From this abstract model of the Service, it is possible to extract interface definitions (in WSDL, for example) and data definitions (in XML, for instance) to represent it as a Web-Service which can be implemented. The Service User who invokes the Service and the Service Provider who makes it available need a shared understanding of a common business process for the Service, with a shared formal model of the behaviour and data (Fig. 3.7). In applications, this is realized with an Interface to access a Web-Service that has commonly agreed operation definitions (WSDL) and data structures (XML schemas).

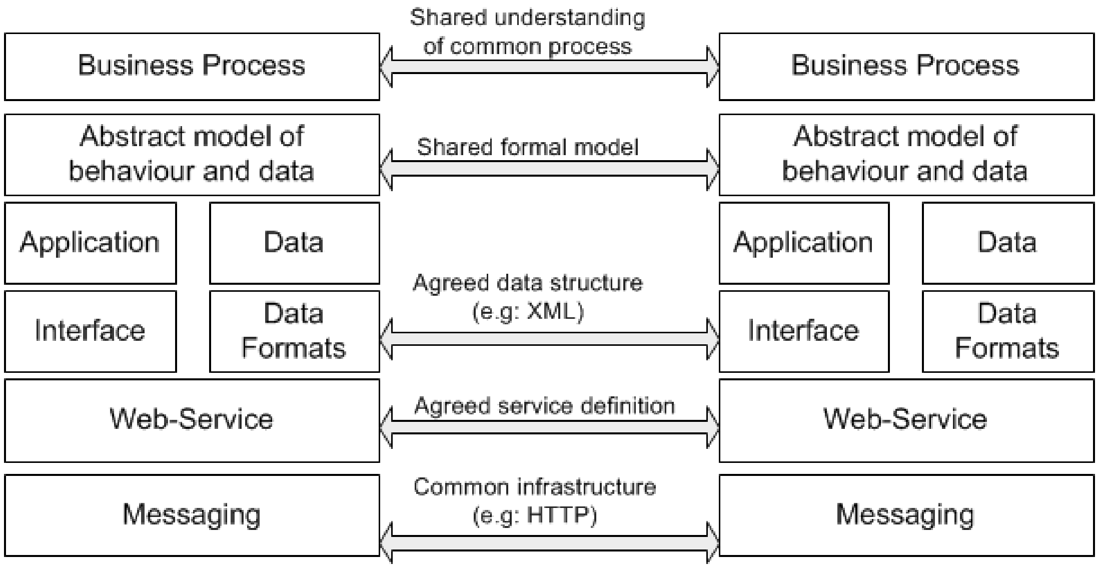

Business Process
Shared understanding of common process
Business Process
Abstract model of behaviour and data
Shared formal model
Abstract model of behaviour and data
Application
Data
Application
Data
Interface
Data Formats
Agreed data structure (e.g: XML)
Interface
Data Formats
Web-Service
Agreed service definition
Web-Service
Messaging
Common infrastructure (e.g: HTTP)
Messaging

FIGURE 3.7: Integration of Business partners [4]

The Web-Service implemented according to the standard model can communicate with nodes across a shared Messaging infrastructure using standard and open specifications like SOAP or REST. It is necessary for two collaborating systems to agree on a range of specifications at different degrees of abstraction to successfully communicate. However, some aspects of the systems may remain proprietary to each partner, including the data, the application code, and the interface (API) used to integrate the application code with the Web-Service.

Numerous Service description schemes are available today besides the ones based on the combination of WSDL/XML/UDDI/SOAP [126]. Some of these description schemes can be differentiated based on the Domain they are used in like communication, e-commerce, scientific-computing etc. Ontology based schemes such as OWL and OWL-S [128], can be used to identify how domain Services relate to each other. But these approaches do not how the data model (encoded in different schemes) is to be implemented or shared across services. It is unlikely that a single service representation scheme will be adopted, and many domain specific extensions are likely within existing schemes.

Hence, a future proof approach should not necessitate the use of a single representation scheme but allow various domains that employ different scheme for representing services co-exist.

3.3.2 Partitioning the Service Space

Partitioning the domain functionality (Service space) into Service boundaries is a domain knowledge dependent exercise. Certain principles need to be adhered to avoid Service Boundary overlap and functionality duplication in the Service Inventory. While each domain can apply varying principles to accomplish this exercise, we propose the following main tenets for a communication domain. Each Service boundary should include considerations to accommodate preferably all of tenets, especially the Utility Services. Higher level Services like Entity and Task Services can use these as recommendations or pointers, since the reusability potential is low and is more focussed on particular functionality. The main partitioning arguments are:

Functionality is the main criteria behind partitioning the Service space is functionality. Each well defined and required functionality is implemented availed as a Service and constrained by the Service boundary.

Reusability is the ability of the Services to be used in various composition. The most basic services within the communication domain are Utility services and are the most reused[1].

Autonomy implies that Services should offer a clearly defined functionality with minimal dependencies on other external Services. This increases reliability and Modifiability of the Services themselves and increases confidence in composite service composed out of such component Services.

[1]See section 3.3 for Service classifications

Tussle defines the ability for service boundaries to contain *Tussle* [7]. For instance, even if it is possible to classify some functionality as a single service, there might be a change that that functionality depends on two or more component functions that evolve independently under conflicts. It makes better design sense to apply the service boundary at the component functionality to let the Service evolve separately[1].

These above mentioned arguments are used in conjunction with already existing SOA Service attributes like discoverability, composability etc which are covered later in this chapter.

Distributed systems approaches such as e-flow[129], StarWSCOP[130] and Computational Grid[131] already use Service Oriented approaches to abstract computational resources as Services. Our proposal integrates such approaches with the networked communication domain by including communication resources as Services as well. The outcome is a communication network architecture that treats distributed computational resources and communication resources as a uniform abstraction of Services, which can be invoked in a similar manner. The application, thus need only worry about functionalities which they need from Services, without having to delve any deeper.

From various studies applying SOA to distributed computing, the services can assume various *Roles* [126]. From our point of view, there can be two specialized roles for services:

Service Consumer *requests* functionality from another Service.

Service Provider makes available Services to be invoked by other Services.

[1]For example, it is easier to visualize a component service that implements the AAA (Authentication, Authorization and Accounting) functionality, but each of these evolve separately under tussle. The ideal design choice here is to have three separate Utility Services that accomplish each functionality separately. Domains can offer aggregated AAA services as a composition of these individual component services

The Service Consumer invokes, initiates and terminates the required Service(s) and handles exceptions generated by the invoked Services. The Service Provider offers an interface for invoking a service and other related criteria, along with specification of parameters associated with managing the service. The network architecture can then be built purely based on the aggregation, decomposition, discovery, composition and execution of suitable services.

There are various Design Patterns that can be leveraged to partition the service space and apply the boundaries. These patterns derived from Service Oriented Design [132] and Distributed Computing [126] enable us to provide proven solutions to common design problems. A catalog of design patterns established with in SOA is covered in [132][1]. Apart from the patterns from SOA, the following concepts are adopted from Distributed Computing paradigm into our architecture:

Service Broker is used as an intermediary between a Service User and a Service Provider,and can be used to discover and synchronize Services based on one or more criteria, dynamic Service selection or support registration of Services (or the above functionality combined together for a *matchmaking* Service to locate a Service of interest). We utilize this pattern to provide common intermediate Services of varying complexity to tradeoff between reusable simplicity in Services and a usable higher level function[2].

Service Adapter is used to enable a Service Provider 'wrapt' a non-standard functionality (an application or a library), and make this available as a standard single Service. This approach can be used to interoperate with legacy functionality and other non-compatible Services.

Service Decomposers are specialized brokers that decomposes a service request to sub-requests to find better matches for Services, in case a single Service that satisfies the request is not available.

[1]A descriptive take on how Web-Services are used to implement the *Decoupled Contract* in SOA is covered in [132]

[2]A Broker Service may use a number of other services to achieve its objective.

Service Composers ,like Service Decomposers, are specialized broker Services which aggregates components service functionalities into a single Service in response to a request from the User.

Discovery Service addresses the most significant part of the process, to find a match between the Service request and the Service provided by a Service Provider. Various sub-patterns on how the matching is done (based on syntax, context, semantic)[132] are available, but beyond the scope of this report.

Service Optimizer maybe used to improve the performance of s group of Services by sharing common request, advance reservation of Service(s), or caching resources for pre-composed frequently used Services.

Reputation Service is a pattern used to include trust, risk and related assessments (such as QoS certification) to increase the reliability of the architecture. This pattern can be applied to produce numerous Services which handles various subcategories in the trust generation, risk mitigation and maintenance.

Execution Platform implies a middleware which supports the launching of Services on available computational resources and monitor their life-cycle. It is necessary that several Services need to be initiated and coordinated to achieve a certain functionality. It is overwhelming for one application to manage such execution, due to complexity and diversity. This responsibility is delegated to an Execution Platform (Like SEA) to act as an intermediary to resource management systems[1][133, 134].

Service Migration pattern is used to support Mobile Service not tied to a particular host (or domain) which may be migrated on demand. This pattern also support state migration of Services to a remote location to create a new instance of the Service.

[1]Where a tight coupling between the User Service and the communication resources are required, execution may be directly managed by the User Service.

3.4 Service Design Principles

One of the ultimate goals for using Service Oriented Architecture is to minimize the amount of custom development effort required to implement solution and instead reuse more and more of the available services to this end. Development of these Services should follow a specific approach to design (Service Design Principles) agnostic services at the utility and if possible, entity service level [5]. Agnostic Services imply that its logic is independent from its business process, proprietary technology or application platforms and thus have high reuse potential as the generic logic implemented can be used in multiple compositions. [1]

3.4.1 Service Contracts

Standardization plays a significant role in Service definitions and agreements to ensure that the components selected by the Service Users form a various sources work together. Without standards, the scope of the definitions his ad-hoc and becomes unsustainable in a wider sense. Standardization is not only required in data interchange specifications, but also in behavioral modeling. The functional service expression, the data formats and associated agreements setup the *Service Contract* where standardization is inevitable to provide interoperability and reusability. . They form a core part of the Service design. Depending on the context, contracts can have various definition. With in the context of Service Oriented Architecture, Erl [5] defines it as:

> "A contract for a service (or a *service contract*) establishes the terms of engagement, providing technical constraints and requirements as well as any semantic information the service owner wishes to make public."

[1]For a lucid explanation of the various service design principles and how they influence each other, see Part II of [5]

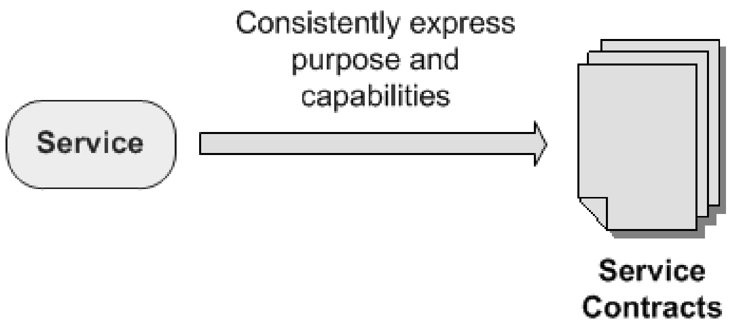
Comprised of one or more service description documents, describing technical interface and other relevant (maybe non technical information). An example of such non-technical service description is the Service Level Agreement (SLA), which can be used to establish the QoS characteristics, such as availability, accessibility and performance. Such requirements, while not technically binding might have legal implications and is hence important.

Services share standardized contracts to enable the interoperability and to allow the capability and purpose of the service to be easily understood. Each individual parts of the contract need to be carefully designed since they expose the capabilities of the services and restrict how other services can make use of them.

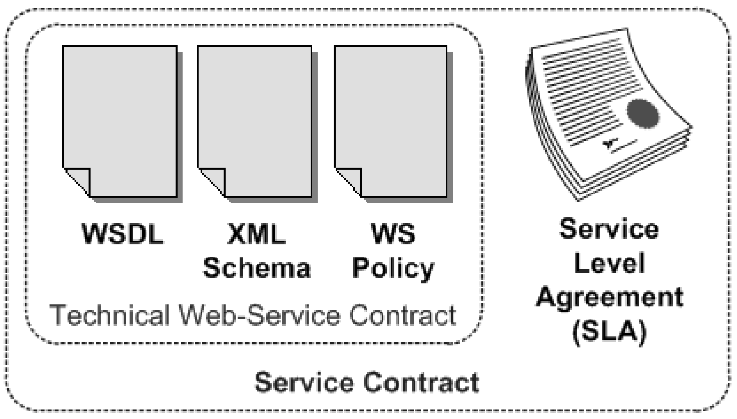
Normally, technical contracts have been specified using interfaces such as Application Programming Interfaces (APIs). These APIs can be accessed remotely (to invoke the service through the network, for example) through Remote Invocation Frameworks like Remote Procedure Calls (RPCs). These frameworks are frequently expressed using Interface Definition Language (IDL) and the Abstract Syntax Notation (ASN).

These APIs were very platform specific and proprietary, and were often created as a part of the custom designed solutions. This Web Services principle establishes the requirement for a non-proprietary distributed communications framework with WSDL as the core part of the technical service contract.

The technical interface description forms the basis of service contracts. This description, most often will contain a formal definition of the data (input/output) required by each service capability. If two service capabilities within the same composition represent the same data using different representations like schema and data models, this follows that the data representation is not standardized. This scenario can lead to incompatibilities mitigated by using data transformation. Besides, the data transformation logic needs to implemented as an actual software component (such as an XSLT style sheet) and runs every time the incompatible Services needs to be access/exchange information, which is inefficient and avoidable using standardization. In Web Services, data representation



(a) Visualizing Service Contracts



(b) Service contract components in Web Services

FIGURE 3.8: Service Contracts

architecture can be developed and standardized independently from service layer and allows for centralized schemas (which can be shared). However, SOA does not require *global* data model standardization, since Domain Inventory design pattern supports partitioning of an enterprise into separate domains which can be separately standardized.In this pattern, the main areas of standardization are functional expression, data representation, and policies. Naming conventions and policy standardizations are also required for interoperability[1].

[1]For a more detailed discussion on the implications and significance of this design principle, see Chapter 6 of [5]

3.4.2 Service Coupling

Coupling indicates the dependency among Services and is unavoidable. Anything that connects has coupling and coupled things can form dependencies on each other. A measure of coupling between two services is the level of dependency that exists between them [5]. For instance, depending on the directionality of the coupling it could be bidirectional or unidirectional. This pattern focusses on the dependency between the services within our framework. One basic approach when specifying Services is that they must be loosely coupled from each other with their service contracts having as much independency from their implementations. The service contract is the core element around which most coupling-related considerations revolve.

The various dependencies which lead to service contract coupling (in various degrees) can be from service logic, vendor technologies, implementation technologies, parent business processes and other services which will be used during composition. However, to avoid further undesirable dependencies, consumers of the service should be restricted to accessing the capabilities of the service via the published contract only. This, while creating a service to consumer coupling will centralize the logic design pattern.

It is possible for consumer services to have *indirect* coupling to underlying technologies and implementations via the tightly coupled Service Contracts of the consumed service. This can be minimized by minimizing the (undesirable) contract coupling during design time (Figure 3.9). The indirect coupling and passing negative coupling characteristics being passed on through chains of services arises as issues during service compositions. Therefore, individual attention to each inter-Service coupling is needed during design time. The Service model (discussed earlier in section 3.3) also serves a model to indicate the level of coupling they exhibit. Task services are sometimes functionally coupled, business logic coupled or even service-to-consumer coupled while Entity services provide the opportunity to implement a independent and decoupled service contracts. Utility services are usually implementation coupled or sometimes vendor technology coupled.

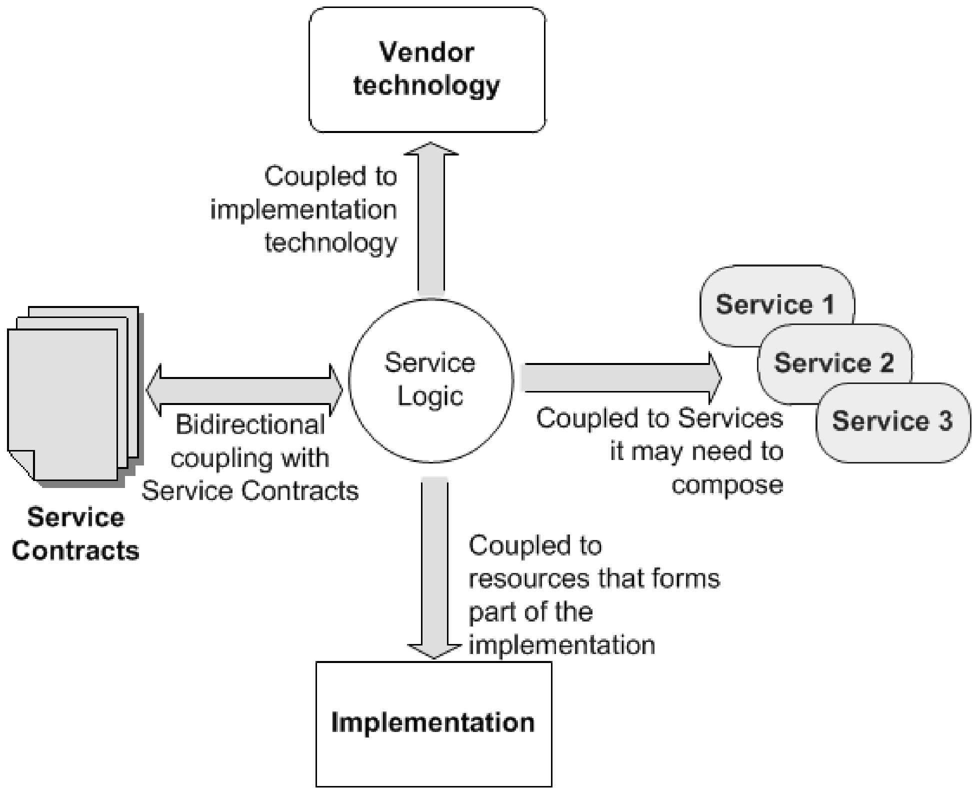

Vendor technology
Coupled to implementation technology
Service Logic
Service 1
Service 2
Service 3
Bidirectional coupling with Service Contracts
Service Contracts
Coupled to Services it may need to compose
Coupled to resources that forms part of the implementation
Implementation

FIGURE 3.9: Service coupling visualization

Within a new framework, factoring the Services and specifying the Service boundaries should be specified with this pattern in mind.

3.4.3 Service Abstraction

Service Abstraction indicates the process of hiding information about the service not necessary for the service to be used effectively. The abstraction of legacy functionality, technology and implementation (i.e, non-essential service information for service consumers) reduce the impact of migration and increase the level of interoperability and reuse.

3.4.4 Service Reusability

This design principle necessities that the Service is defined by context that is agnostic to any particular scenario and that the logic should be generic enough to facilitate numerous usage scenarios. Besides, the Service contract should be generic and extensible. Service could be reused by the same consumer service repeatedly to fulfil a single business task or it can be invoked by numerous consumer services as a part of composition to solve multiple tasks.

3.4.5 Service Autonomy

Service autonomy indicates the independence of the service implementation. Highly autonomous services will have higher ability to evolve independently from external influences, and will be more reliable. If the Services are designed to be autonomous, they will exhibit consistently reliable runtime execution performance, greater degree of performance reliability, an option to be isolated in response to specific security, reliability or performance requirements, among other characteristics. Besides these advantage, compartmentalizing Services through Service autonomy has Security implications as well. Autonomy allows isolation of Services in response to a change in policies or invoke alternate behavior based on policies.

Another significance for this design principle - Composition of services are inherently non-autonomous process and produce a task service whose autonomy depends on the collective autonomy of the component services. Thereby, the reliability of the composed service depends indirectly on the autonomy of individual component services. As is evident from definitions, Service autonomy and service loose coupling have a close relationship. Autonomy can be *runtime* autonomy or *design time* autonomy, where former implies the amount of control the service itself has over its execution environment at runtime and the latter indicates the amount of governance control a service owner has over service design. A rudimentary but expandable measurement framework for autonomy, classified into contract autonomy and implementation autonomy can be found in Section 10.4 in [5]

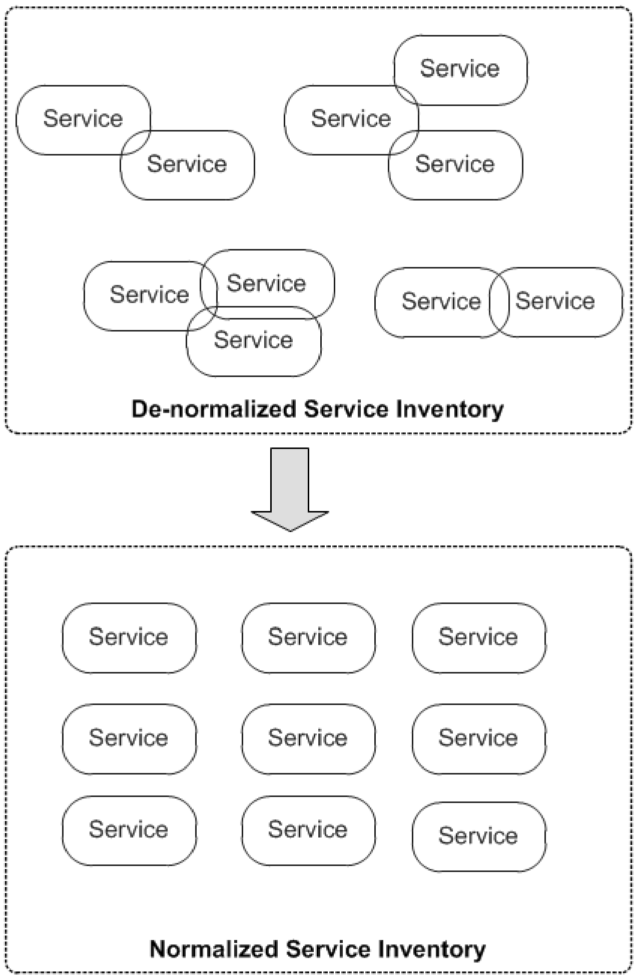


FIGURE 3.10: Service Normalization[5]

3.4.6 Service Statelessness

The state of a Service implies its general condition. During the invocation of a service, certain data might be generated in relation to the functions provided by the service. The amount of data that needs to be retained during the lifetime of a composition can be substantial. The effort to manage such generated data over the lifetime of the service activity, even if it is temporary, (or *state management*) can have notable effect on the performance of the architecture. The performance bottleneck due to the state memory can be significant for a service.

Service can be active or passive in state, while the state conditions can vary on Stateless. The Service state depends on the amount of data retained or processed during various stages of its lifetime. State related data could be Session data, Context data or Business data. The design priority should be to maximize Service statelessness.

3.4.7 Service Discoverability

The process of searching for and finding solution logic within a specified environment is referred to as a discovery. To accomplish this it is necessary that information about services, their capabilities and policies be consistently, clearly and accurately made available to be discovered. These are usually the metadata relevant to the services and includes the functional capabilities (from standardized Service Contracts), relevant technology and programmatic information, and QoS specifications. The QoS here implies the behavioral characteristics, operational thresholds and policies runtime of the service. Thus the metadata describes the purpose, capabilities and limitations of a Service.

This information (metadata) should be stored centrally (ideally) and be accessible in a consistent format so that users can locate, retrieve and interpret services in a simple and standard manner. This central location (physical/virtual) where a user can find the relevant services is termed *Service Registries*. UDDI [100] is a such popular open standards based registry. One important property of such registries is that they should be *searchable*. This facilitates the short listing of suitable solution logic based on keywords or tags. The central role played by this principle is evident from the traditional *SOA Triangle* model.

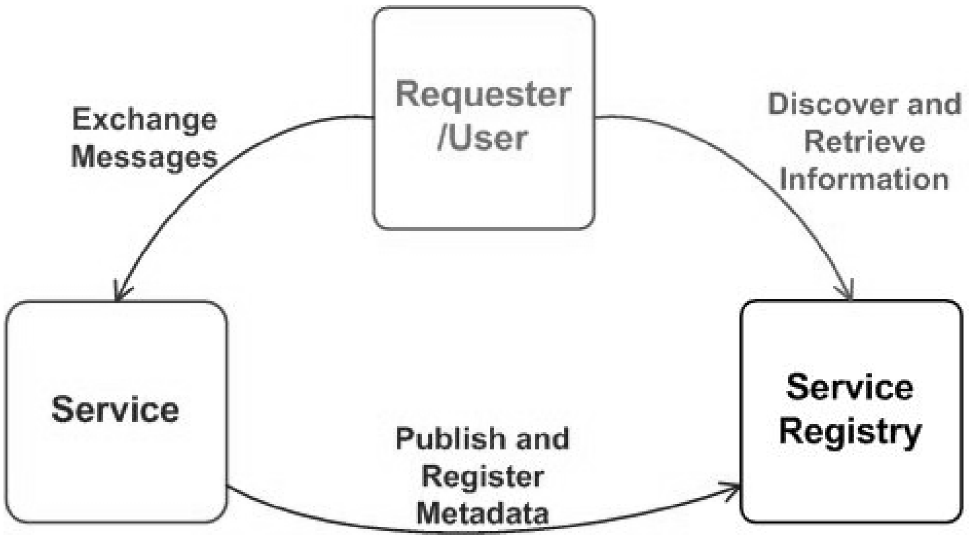

Requester /User
Exchange Messages
Discover and Retrieve Information
Service
Service Registry
Publish and Register Metadata

FIGURE 3.11: The traditional SOA triangle model, indicating the central nature of the Service Discovery Facility

This design principle significantly impacts the reusability factor of the parent service and normalization of service repositories. The lack of discoverability can also add redundancy and governance overhead to repositories through design phase and persist during runtime. The quality of metadata associated with a service is an indication of its discoverability. This principle has a significant impact within the context of our architecture. Adding semantic description of domain Services to Web-Services using Ontologies can result is a much more descriptive version of Web-Services (called Semantic Web-Services). Adding semantic information to WS increase the scope and precision of Service discoverability [135].

3.4.8 Service Composability

The Service Orientation approach provides a design platform where logic is decomposed and recomposed [5]. This is a natural fit for Distributed Computing paradigm where assembling capabilities from different sources to solve a larger problem forms the foundation. Composability is one of the key design principles which results from the correct application of all the other design principles mentioned above. All the other design principles in the Service Oriented paradigm directly or indirectly supports the extend of composability attainable by a Service [5]. This design principle states that Services are designed to be composable.

Depending on the complexity of composition and the size of service activity, Compositions can be classified into simple and complex, although the demarcation between the two are often blurred and not that evident. One example of a simple composition would be a hop-by-hop routing between nodes (across multiple nodes, even) where processing is based on a single document or a few parameters. A business task however could be complex depending on the number of messages exchanges and services involved.

3.4.9 Service Inventory Blueprint

The aim of the Service inventory blueprint is to provide a complete perspective of solution logic across a specific domain represented by an inventory of service candidates. Each Service boundary must be modeled to accurately represent the service's functional context while not overlapping with other service boundaries. Also, the dependencies among services should be mapped to provide a clear picture on the type and quantity of logic each should encapsulate.

The service candidates established within the Service inventory blueprint form the basis Service contracts with well-defined functional scope. The steps involved in creating the inventory blueprint are [5]:

1. Initial population of the Inventory (possible from domain research, or Service delivery projects)
2. Hybrid applications and a growing Inventory. More legacy applications wrapped in standard interfaces become available for migration and backward compatibility.

3. A mature Service Inventory where majority of the new tasks encountered can be addressed using compositions of services from within the Inventory without modifications.

3.4.10 Service Normalization

Normalization is an approach to reduce or even eliminate redundancy across data entities and structures. In the world of service-orientation there is a specific Service Normalization pattern is applied to minimize the amount of functional redundancy across a service inventory and is one of the primary reasons to invest in the creation of a service inventory blueprint prior to actually building the service inventory. This principle tries to avoid functional overlap which can lead to redundancy in a service inventory, resulting in functional denormalization and potentially convoluted composition architectures.

The maintenance of coherent Service Inventory becomes an issue when Services within the Inventory needs to be changed (from implementation itself to parts of the service description in a service repository) due to various reasons. Although modifications to an existing Service can be done by extending its capabilities while maintaining backward compatibility, it is not always possible to accommodate all future requirements this way. Hence, new *versions* of the Service will need to be introduced, which introduces certain challenges, such as versioning. There are many approaches to Service versioning. One popular approach is to replicate previous versions of the Service and add additional or modified elements to form the new Service. New versions are named differently (by using some naming convention), and their description is stored in the registry as a new entry. Since WSDL and UDDI do not inherently support versioning, Juric *et al* [136] propose extensions to WSDL and UDDI for service versioning. The approach addresses run-time and development-time versioning. Notifications about new and deprecated versions are communicated to Service Users with traceability support to track changes.

Service Discovery design principles are significant for our architecture at various levels. Traditionally, Services were described and documented to be discovered and interpreted by humans. Subject matter experts and system architects could search a given system registry or database (even a simple webpage, table, excel sheet or LDAP database would suffice) and interpret the metadata and related documents to conclude if any of the available services are appropriate for the task at hand. This approach (commonly termed *Design time discovery*) might work for enterprises with limited scope and scale. However, it is imperative to have automated composition capabilities for dynamic environments like networks. This highlights one of the challenges faced by Web Services today. To have automated composition, software agents should be able to *discover*, *retrieve* and *interpret* services on their own with minimal external influence. This follows that not only service metadata should be described in rich syntaxes, but also the service capabilities should be expressed in machine understandable manner (probably semantically) so that they can be modeled by software. This state of the art in technology today does not make it possible for this to happen reliably. Dynamic discovery and automated composition of Web Services is an ongoing research topic and the outcome of that studies will have significant impact on some of the core ideas proposed in this thesis. That being said, it is possible to have limited scope automatic composition today by specifying and restricting the process flows of composition behavior to a subset of possible predictable behaviors.

3.5 Service Composition

Web-Services are sufficient for most of the simple interaction needs, but they might not suffice for integration of unique process requests that involve multiple Services. Business process integration, for example, in real business scenarios involve long-running interactions, transactions management, stateful invocations and are often driven by a workflow engine that execute a specified business process model to automate the information flow and the business operations. This raises the needs for coordinated aggregation of

Web-Services that provides the mechanism to fulfill the complexity of such processes. The process of developing a composite service is called *Service Composition*[87].

There are two main research trends to realize Service Compositions. Among the business community, Service specification, composition and execution standards revolve around XML syntax (for example, BPEL, WSFL). Another approach is by by Semantic web communities (mostly academic driven) focus on reasoning based on Service preconditions and effects held within according ontologies [90, 137].

In the latter approach, Services are regarded as network resources described in Resource Description Format (RDF) [138] using RDFSchema [139] and other Semantically expressive languages. One approach is to enhance Service descriptions with semantics such as inference and reasoning mechanisms by extending existing WSDL annotating (with inputs, outputs, preconditions, and effects, for example) to produce WSDL-S [140]. Similarly, OWL-S [128] uses the Web Ontology Language (OWL) for semantic descriptions and also provides a WSDL grounding to execute Web Services. Both the Semantic Web Services Framework (SWSF) [141] and the Web Service Modeling Ontology (WSMO) [142] initiative define a description language, ontology, and rules to provide a framework for multiple tasks in the Semantic Web Services domain such as description, execution, or reasoning. METEOR-S [142] is another approach for the semantic service annotation with specific focus on QoS and composition features.

To accomplish the fusion of single Services, plans[1] are required which contain the abstract specification of a set of services together with specific control structures defining their interaction and data flow. Within Service Composition plans, Services are regarded as blueprints, i.e., as placeholders built up of semantic descriptions that specify the type of service that is required. During composition, these blueprints are bound to real services, providing an instance of a service composition plan [143].

The semantics of Web services is crucial to enabling automatic service composition. It is important to ensure that selected services for composition offer the right features which

[1]often called Service Composition Plans

may be expressed syntactically or semantically. Semantic features of Web-Services are expressed using ontologies [144] and play a central role in the Semantic Web, extending syntactic service interoperability to semantic interoperability [145]. An ontology is a shared conceptualization based on the semantic proximity of terms in a specific domain of interest [135]. Several techniques are proposed for composition and recently automated composition based on semantic web technologies have been becoming prominent. For example, Ramparany *et al* [146] proposes a composition technique for device services based on abstract composition plan considering additional information like contexts. However, these kind of approaches need the extension of standards semantic languages (in this case, OWL-S) to accommodate attributes like type, inputs, outputs, local parameters, input and output properties, as well as a context condition.

The common pattern used within Service Oriented approaches to determine the requirement for Composition are termed Service *Aggregator/Decomposer*. These can utilize domain specific ontologies to determine alternative service providers of interest, and enable these requests to be forwarded. The approaches adopted depend on the representation scheme used to encode the ontology, and the service definitions. The aggregator/decomposer must use the same representation scheme as the service discovery agents and confirmed through an initial message exchange. This pattern provides a means to split a service request into sub-services, and to subsequently compose the results. The Aggregator/Decomposer pattern is applicable when a large number of computational services exist, but there are no requests which exactly match these services. Also, if no suitable computational services can be found, or if the computational services most suited to running an application are busy or computational services which match a particular request criteria (such as cost, performance or security/access rights) cannot be met, this pattern can be applied. This pattern is also useful in scenarios where the Service User may wish to tradeoff criteria such as precision vs. speed, precision vs. cost, or performance vs. cost among available Services.

3.5.1 Service Composition Challenges

Automatic service composition is a critical requirement[135, 137]. Automation of service composition requires not just functionality of inputs of services, but also much more metadata such as service model, preconditions (input state and input data) and postconditions/effects (output state and output data). Automatic composition implies that a method can generate the process model automatically and the method can locate the correct services if an abstract process model is given [147].

One of the main problems faced during Service composition is the non-standard naming conventions used for Services [148]. Other issues identified with automated Service composition include Service coordination, Transactions, Context, Conversation modeling, Execution monitoring, Infrastructure which are covered in the survey by Dustdar and Schreiner[149]. These are not the only problems limited the widespread use of such techniques. When composing services with in complex environments, coordination of the sequence of events is need to mitigate inconsistency and ensure correctness. Various solutions have been proposed to accommodate this requirement like WS-Coordination [150] and Web Services Composite Application Framework (WS-CAF) [151]. To provide reliability and guarantee to interactions, whether they are atomic transactions or long running transactions, the coordination framework should provide relevant provisions [1]. WS-Transactions[152] is one such specification.

Context implies the *Information utilized by the Web-Service to adjust execution and output to provide the client with a customized and personalized behavior* [153]. The concept of context-awareness seems to be a promising solution for a lot of problems which have been implied by the usage of mobile terminals in ever-changing environments [154]. This property, defined in an extensible manner, can be widely utilized throughout the Service Oriented workflow to efficiently tailor and tweak the logic. WS-Context [155] specifies contexts and its sharing and management.

[1]These protocols should impart reliability guarantees such as ACID (Atomicity, Consistency, Isolation and Durability) to the transactions; These properties might be difficult to guarantee for long running Web-Service transactions

Conversation modeling or composition modeling indicates the composition workflow of the solution from individual services. It includes service discovery, binding (preferably dynamic), service composition model generation and validation, analysis and verification of generated compositions. numerous approaches have been suggested to support defining service models and richer WS abstractions like Web Services Conversation Language (WSCL), Web Service Choreography Interface(WSCI) etc.

Once the composition has been created, verified and validated, it needs to be executed. Here, execution monitoring ensures correctness of the process. Composed Web-Services can be executed either centrally or in a distributed manner. In centralized, a single coordinator controls the execution of the components. In distributed, the WSes execute independently of other participating WSes, but coordinates their activities via a shared execution context. Each domain running a WS has its own coordinator which has to collaborate with coordinators of other hosts to guarantee a correct ordered execution of the services.

Besides the obvious requirements and properties, there can be numerous other factors which can influence the service selections, composition and execution. These could be runtime QoS parameters based on the service environment like scalability, capacity, performance related properties (response time, latency, throughput), reliability, availability, flexibility, exception handling, accuracy etc. Transaction supported QoS like regulatory, supported standards, stability, cost and completeness can also come into play as non-technical properties. Security related like authentication, authorization, confidentiality, traceability, auditability, data-encryption and non-repudiation can also factor in when considering critical services. This implies that a simple registry of services is not enough, since service discovery using metadata may not be enough for automated compositions. One proposal is to implement a QoS certifier [156] as an extension to the service registry. This approach provides a facility to avoid services that are incomplete or broken by verifying the Service Provider QoS claims via the QoS Certificate [1].

[1]The QoS claims by the Service Provider can be verified and the service certified before the registration of a service in the service registry

This requires that current registry specifications (like UDDI) need to be extended with additional information to accommodate the QoS parameters.

Another challenge in the automatic composition of Web services is whether those services are composable [149]. Composability refers to the process of checking if Web services to be composed can actually interact with each other, i.e, that is, if the associated message interchange protocol among them is compatible. This requirement means that the message syntax and semantics should be compatible and deadlock-free. When more and more services are offered and advertised in repositories, there are more chances of satisfying a service demand by composing existing services. However, mediation at the protocol level (using a Mediator component) might be required to solve matchmaking conflicts at the message/conversation level [157].

3.5.2 Orchestration and Choreography

But, Web-Services composition is a highly complex task especially when the number of services increases. Manual processing of all workflow is not a practical approach. Composition rules deal with how different services are composed into a coherent higher level service. In particular, they specify the order in which services are invoked, and the conditions under which a certain service may or may not be invoked. This composition could be static or dynamic [158]. Two possible approaches for the static service composition called Orchestration and Choreography are normally studied [123].

Orchestration and choreography describe two approaches to create business processes from composite Web services. The definitions of these terms tend to overlap but Peltz [123] illustrates their high level relationship as indicated in Fig. 3.12. Orchestration combines available services adding a central coordinator (the orchestrator) which is responsible for invoking and combining the single sub-activities. The interactions occur at the message level and include business logic and task execution order spanning applications and organizations to define a long-lived, transactional, multistep process model.

Orchestration always represents control from one party's perspective. This differs from choreography, which is more collaborative and allows each involved party to describe its part in the interaction. Web services choreography defines complex tasks via the definition of the conversation that should be undertaken by each participant. Following this approach, the overall activity is achieved as the composition of peer-to-peer interactions among the collaborating services.

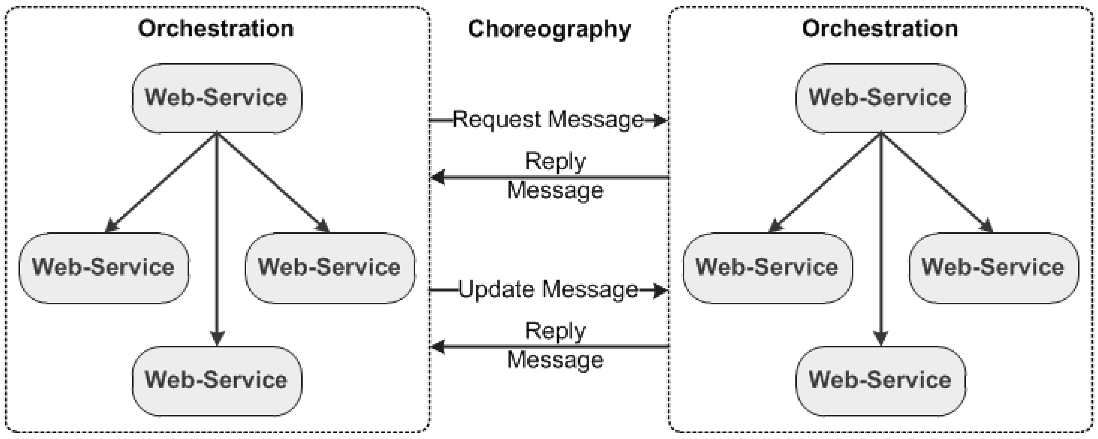


FIGURE 3.12: Service Orchestration and Choreography

Various business process language facilitate Orchestration and Choreography of business processes. BPEL, WSCI, and BPML all take somewhat different approaches to orchestration and choreography [123, 159]. Both BPEL and BPML provide capabilities to define an executable business process, whereas WSCI introduces a collaborative extension to WSDL describing how to choreograph the available WSDL operations. It supports message correlation, sequencing rules, exception handling, transactions and dynamic collaboration. WSCI's approach is more choreographed and collaborative, requiring each message-exchange participant to define a WSCI interface. In WSCI, no single process manages the interaction via basic and structured activities. Other specification include Web Services Flow Language (WSFL)[160], Web Services Choreography Description Language (WSCDL)[161] etc. Most of the standards focus on representing service compositions where the flow of the process and bindings between the services are known *a priori*. A comparison of BPEL4WS, BPML, WS-CDL, WSCI and DAML-S composition languages are covered in [162]. It can be seen that none of these methods support agreements on QoS support in a business to business scenario.

3.5.3 Service Composition Requirements

There are many approaches to Service Composition based on various criteria. A main classification is based on the stage at which the composition is done - static vs dynamic. Static composition implies that the composition is chosen, linked and deployed at deign time. The shortcoming, as evident, is that this approach is too restrictive for updating older Services, Service definition changes and system redesigns. Dynamic composition, in contrast, processes the specification that automatically configure at runtime. The composition methods can also be classified depending on the basis for composition. Compositions could be Model Driven, Declarative [163], Work flow based[164] or based on AI planning[147].

Service compositions can also be classified based on the entity who chooses the composition. It could be manual (done by a human controller) or automated. Automatic composition generation is a challenge with full automation of dynamic composition being still a subject of ongoing research [162]. Dynamic and automated compositions could be possible overcome using semantic web technologies and a survey of dynamic composition approaches are covered in [165].

Service composition will work only with frameworks designed for it. this contradicts the vision of interoperability of Web-Services. The lack of a uniform model or framework by itself is a serious concern to create service composition frameworks, since current specifications lack additional specifications/capabilities to address sematic extension (in WSDL for example). Some composition methods have developed their own data structure for extending WSDL (MAIS for context based, Onto-Mat service and SHOP2 for semantic, WebTransact and StarWSCoP for dynamic composition) which is not an ideal approach and induces fragmentation [149]. Most established methods lack semantic description and composition capabilities.

Component Model

A service composition produces a composite Service from component Services. In a service composition, each component service should be able to be defined in a structured way to identify its nature [93]. As a general requirement, component model should be able to describe the type of component services and service interface. Also, a different but related nomenclature puts *Service selection models* that deals with static and dynamic bindings (design time or runtime).

Orchestration Model

Orchestration describes the way in which different services can be brought together into a coherent composite service to provide a value-added service. Control flows specify the order in which individual operations of services are executed, and the conditions under which a certain service may or may not be invoked [93]. One composition language should provide the control constructs to define logic over a set of service interactions. To support a variety of practical composition requirements, control constructs in many aspects should resemble a programming-language. Specifically, the control construct should be able to break up the flow of execution by employing decision making (if-then-else, switch), looping (while, for), and branching (break, return, continue), enabling the composed service to conditionally execute particular operation. Therefore, control flow is one of the essential requirements for service composition.

This model defines the order and conditions in which the Web-Services are invoked, using process modeling languages like UML activity diagrams, Petri-nets, state charts, Π-calculus etc [149].

Data and Data Flow Model

Service components interact with each other by exchanging data which should be defined and accessed in an explicit ways [93]. From service composition point of view, data flow refers to the flow of information in a composed service. One composition language should provide means to define data, to describe how data can be exchanged between component services being composed, and to describe inputs and outputs of the composed service.

Error Handling and Transactions

It is desirable for a composition system to offer an efficient and effective error handling and compensation mechanism. Such mechanism will help to identify faults (i.e. error handling mechanism) and undo work that is partially or successfully completed (i.e. compensation mechanism). When error occurs during execution, the system can deal with exceptional behavior to guarantee system stability. Transaction models defines which transactional semantics can be associated to the composition. Exception handling models defines handling exceptional states during execution without service being aborted.

Quality of Service (QoS)

Non-functional QoS like timeliness, security, dependability etc. With the increase of Web Services as a business solution to enterprise application integration, the quality of service (QoS) offered by Web Services will become more and more important for service providers and their partners. A better QoS for a Web Service will make it more competitive than others. Therefore, it is desirable to carry the quality of service information which is a key non-functional property for service composition.

3.6 Summary

This chapter took a closer look at the term ‘Service’ and important aspects that need to be taken into account when creating a new Service Oriented Architecture. Especially the way how Services are chosen and described as basic building blocks will influence the composability and orchestration of future applications.

A Service Oriented approach certainly adds flexibility and better application support. On the other hand, many basic mechanisms like dynamic composition, semantics etc are not yet standardized. Therefore, the discussion of a future Internet architecture is important to understand key requirements for the development of SOA mechanisms and standards.

CHAPTER 4

A RELATIONSHIP ORIENTED SERVICE ARCHITECTURE (ROSA)

This chapter explains the proposed architecture and related Services. A description of how the mandatory services fit together to form an implementable network architecture is also given.

Moving away from the layered abstraction, we have identified *Service Orientation* and *Standard Interfaces* as an approach for abstraction. The concept of Services encapsulating functionality provides us with an elegant and flexible concept. What is necessary now is to outline the fundamental set of principles that enables us to tie these Services together in a simple and standardized manner to form a coherent architecture. The functions, which formed the basis for the original Internet becomes functionalities implemented by Services within our architecture. For example, the connectivity across heterogenous networks, one of the basic requirements of IP based networking can be provided by a single service, for example, a *Connectivity Service*.

A reference architecture forms an important step in realizing a reference model and subsequently, a architectural implementation [113]. The reference architecture defines the structure of systems, essential building blocks, their responsibilities and their collaboration. In discussions related to the European perspective of *the future Internet*, Stricker *et al*[166] refers to three types of reference architectures:

Functional Reference Architecture which separates the functionality into logical functional boundaries (or concerns). Their collaboration, the data flow, the responsibilities and dependencies are also specified.

Logical Reference Architecture where the structure of the architecture is defined using components and layers, together with their hierarchy and communication dependencies. No particular implementation technology is specified.

Technical Reference Architecture defines the implementation details of components and refers to specific technology to realize them. Several Technical Reference Architectures can be derived from a single logical Reference Architecture.

In this work, we focus on the Functional Reference Architecture based on Service Oriented approaches. We also refer to the Logical Reference Architecture within a limited scale, where further refining is warranted. Wherever possible, we also indicate the specific technology that can be used to implement the described component or Service, but these do not form a complete technical Reference Architecture. We will specify the Functional Reference Architecture by separating the functional range of the network architecture into logical functional boundaries.

The most important aspect of such an approach is the definition and scope of the services that the architecture presents to applications and to other services with in the domain boundaries. The services that make up the architecture (which are later composed to provide networking as a Service) should be elaborated with definition, functionality, description and design. The principles of service design in SOA is not a new topic [5]. The methods have been tried and tested in various enterprise scenarios over the years.

But, designing services using this methods to cover the Internet as a whole, let alone *the* network architecture of the future warrants much more attention than enterprise services. The loose coupling and abstraction principles [5] which are characteristics of services to be designed within a service oriented framework provide us leeway in this process.

4.1 Principles for Architecture Composition

We codify the design axioms for our new approach in the following list:

- The architecture must be layer-less.
- The abstraction uses Services following a Service Oriented Approach.
- All Services must be handled in a uniform way.
- The architecture must accommodate the concept of administered domain boundaries.
- Accommodate and handle different standards used by different domains.
- Allowing migration towards new Service capabilities and accommodate Service virtualization.
- Allow for transition from past and to future architectures.

We propose a layer-less architecture comprised of Services (Fig. 4.1). The functionality of the Services and their dependencies may vary, but all Services will conform to the same standards of specifying their capabilities and how to invoke them. Services with varying scope, for example Communication Services, Enabling Services or Application Services are all handled equally. A network abstracted as a service or a collection of services has significant implications. Continuing from the previous section, where we discussed the rationale behind our approach, we outline the architecture of our approach

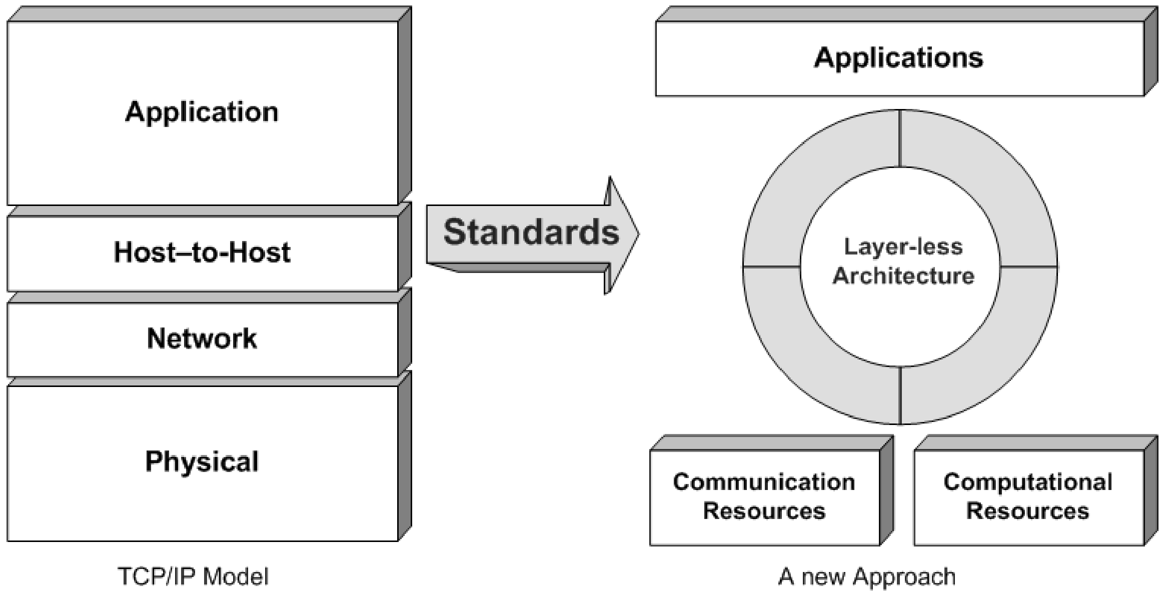


FIGURE 4.1: Network resources abstracted as a Service

in this chapter. While, the approach of *Service Orientation* and *Network as a Service* might not be by themselves new paradigms, our approach of top down abstraction of Network architecture based on Service orientation is unorthodox, and arguably compelling enough proposal to warrant a serious discussion.

Various other approaches using SOA use specific models to compose together individual services, like business rules [167] or predetermined models [149]. These can be restrictive to a more generic approach such as the one proposed in this thesis. Composition based on business rules is ideal for intra-enterprise or inter-business service composition, but this approach requires agreement on the specific business processes and workflow prior to compositions. The model based composition is much more suited to static compositions, where the models are predetermined and not generated on the fly. For a dynamic environment with changing service landscape, a much more dynamic principle needs to be isolated to form the basis for composition. We identify two such principles and argue that these can form the fundamental approach for Services to be brought together to form a usable framework. The principles we propose are *Meeting of Domains* and *The concept of Relationship*.

4.1.1 Visualizing *Meetings of Domains*

When two entities meet (i.e, share a channel capable of exchanging messages) they might wish to interact with each other. The *entity* here do not particularly refer to a node, a device, a piece of software or a service. We define an entity (within our architecture) as anything that offers or consumes a Service and has a well defined boundary with a single point of access or contact. Thus, an entity here could be a Device or a business domain or a composite service etc. The boundary could be physical, ownership, service type etc. We would like to be agnostic regarding how the intelligence for the service is implemented. We represent the wish to communicate of two domains (with the intent and ability to do so) as the *Meeting of Domains* (Fig. 4.3(a)).

Domains can be visualized as a region (physical or virtual) characterized by a common theme and restricted by boundaries. A generic example for such a domain could be a Business Entity with business boundaries, offering specific services and characterised by business processes. This approach has been used elsewhere as TurfNet[1] [168] and Ambient Networks [35], although in different contexts. We foresee administrative (including possibly legal) control as the main criteria that outlines the domain boundary. The functionality offered by a domain is represented as Services within the domain as visualized in Fig. 4.2. Domains gives us a generic enough platform boundary within which the services can reside and interact. For example, a transaction between services of different business entities (B2B) naturally classifies into a meeting of business domains (where the boundary is defined by business ownership). A physical device forms a domain with the common characteristic of a physical boundary and (usually) a single ownership. In this case as well, connecting two devices together can be reduced to meeting of two domains as illustrated in Fig. 4.3.

This *Concept of Domains* can be extended recursively for already collaborating domains, i.e, when two domains meet and agree upon the various aspects of service sharing and usage, their collaboration can be again abstracted further as a single domain, which being

[1]Defined as a 'completely autonomous network domain'.

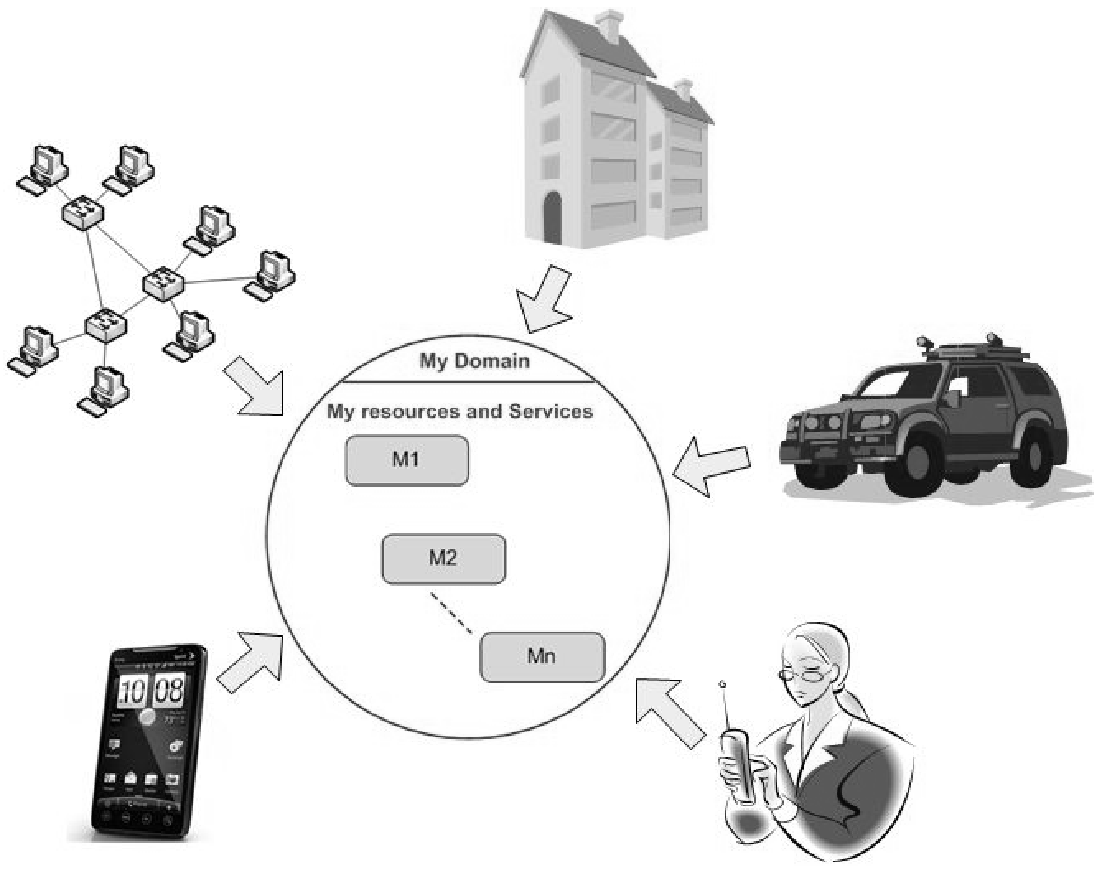
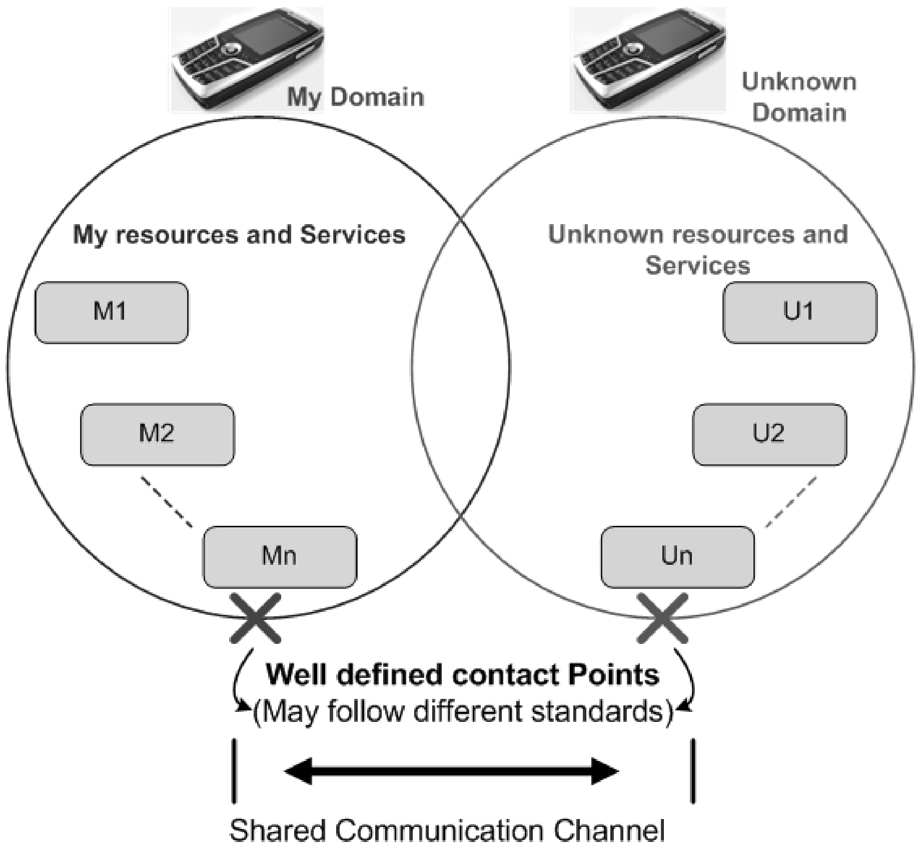


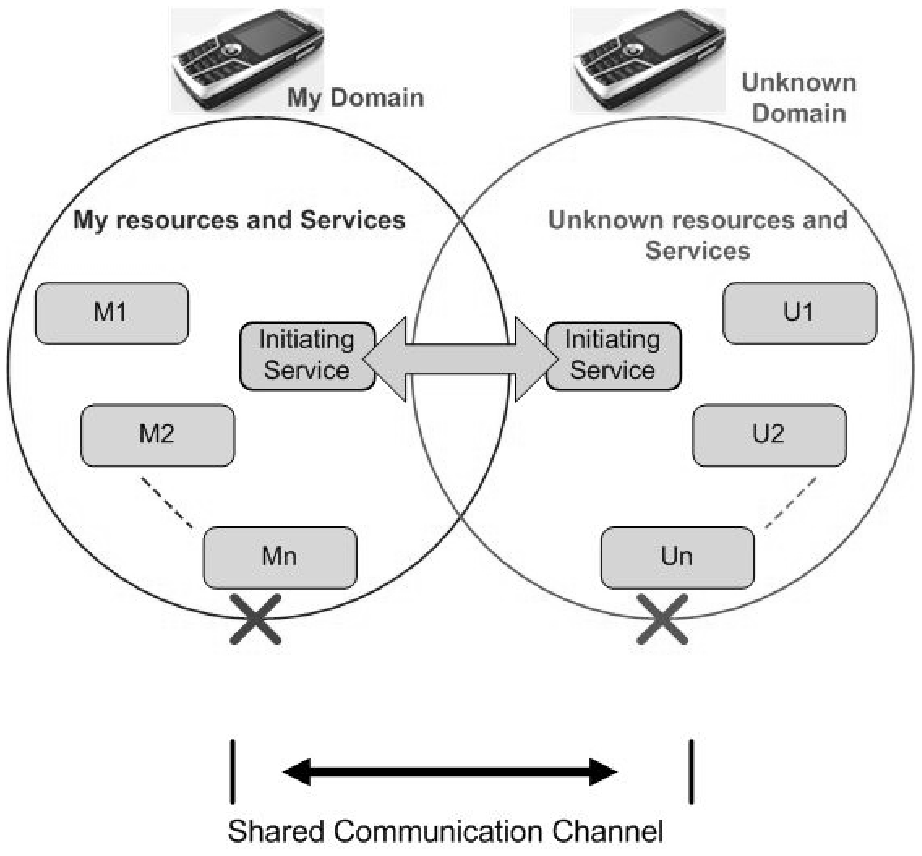
FIGURE 4.2: Reducing administrative regions and functionality into Domains and Services

the collection of capabilities that they together can perform. The implications of such recursive nature derives from the composability of services within an SOA. There are two basic requirements for the meeting of domains view. One of them, as mentioned before, is the necessity of a common communication channel. The other requirement is the presence of a well known entity which speaks a common language to initiate the *bootstrapping* of the communication environment [1]. This entity presents a well know address (to be reachable) and a well described service (to be usable) to form the starting point for further interaction between the domains (see Figure 4.3). This implies that the common point of interaction is a well known service which fits in well within our scope of a service oriented framework. In such a scenario where a well defined contact point (a service) exists, the meeting of domains starts with the exchange of information between these well known services.

[1]This is analogous to various protocols using a *well known* port number to initialize connection


My Domain
Unknown Domain
My resources and Services
Unknown resources and Services
M1
M2
Mn
U1
U2
Un
Well defined contact Points
(May follow different standards)
Shared Communication Channel

(a) Domains meeting


My Domain
Unknown Domain
My resources and Services
Unknown resources and Services
M1
Initiating Service
Initiating Service
U1
M2
U2
Mn
Un
Shared Communication Channel

(b) Abstracting with a Well-known Common Service

FIGURE 4.3: The concept of *Domains Meeting*

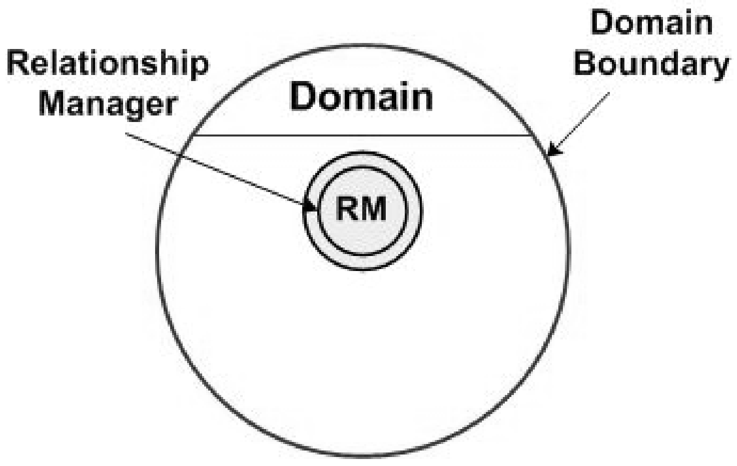


FIGURE 4.4: A Domain and its Relationship Manager(RM)

The *Meeting* of domains can vary in scope like *Merging* of Domains where the meeting domains loose their individual identities and form a new merged domain[1] or *Interconnection* of one domain's capabilities to another domain while preserving individual domain autonomies[2]. The meeting of domains forms the starting point for the central architectural principle of composing a Domain network. Domain Network composition allows communicating Domains to automatically negotiate inter-working agreements and policies using which they establish an inter-domain communication. Our architecture maintains the abstraction through Services as the absolute paradigm and hence do not differentiate control functions from other Services. The 'Relationship Manager' (RM)[3] of a domain (Fig. 4.4) is responsible for the bootstrapping, negotiation and establishment of a relationship with another domain (through it own RM). This indicates that the Relationship Managers of collaborating domains form a control plane, coordinating traditional control functionalities in a network Fig. 4.5. This common common Service replaces the need for high level control Interface for the architecture. The domain itself can be considered as a Service by other domains (or even Services) prior to Domain Network composition, with capabilities published/negotiated by the RM. The domain Services are administered by this coordinating Service, which presents an overview of the overall capability of the domain. Iteratively, a domain like an enterprise domain or

[1]Similar to Horizontal Composition in TurfNet[168]
[2]Analogous to vertical Composition in TurfNet
[3]See subsection 4.3.1 for more details.

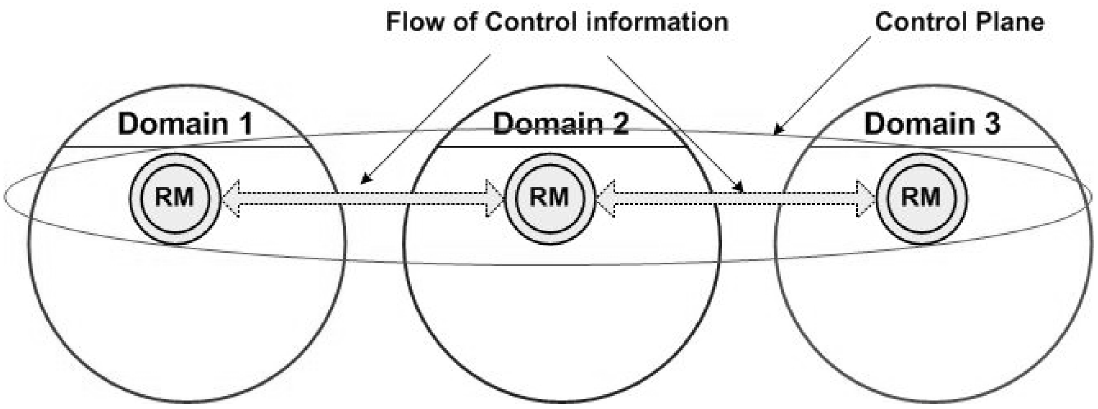


FIGURE 4.5: The control plane indicating flow of control information among RMs

the web or a device can have subdomains delineating certain functionality or administratively delegated control. Such sub-domains are administered by a coordinating sub-function within the scope of that sub-domain. This Service can as a point of contact for that sub-capability or Service. At domain level, these sub-functions are coordinated by the domain level RM. For example, a camera with multiple imaging capabilities (a high resolution camera, a low resolution front facing camera and an IR camera) can present its functionality to other Services via a combined *Imaging Sub-Domain*, accessible as a single imaging Service. A single point for accessing various imaging capabilities is accomplished through the sub-domain RM (RM2 in figure)Fig. 4.6. The imaging sub-Services can provide access to each of these capabilities. These capabilities can be grouped under an imaging Service with each sub-Service encapsulating each capability and coordinated by a sub-function (for that sub-Domain). External Services will still have to negotiate with the Domain RM (RM1 in Fig. 4.6) to access its Services including any imaging Service. This indicates that the mobile device can share its imaging capability (if it allows) with another domain (another mobile device) which might not have the capability. A simple example is to use another Mobile Phone's camera to take pictures.

Conversely, collaborating domains can agree to merge their capabilities as being presented by a single domain with a single point of contact (The collaborating RMs can decide to delegate or replicate their functionality among themselves). Depending on

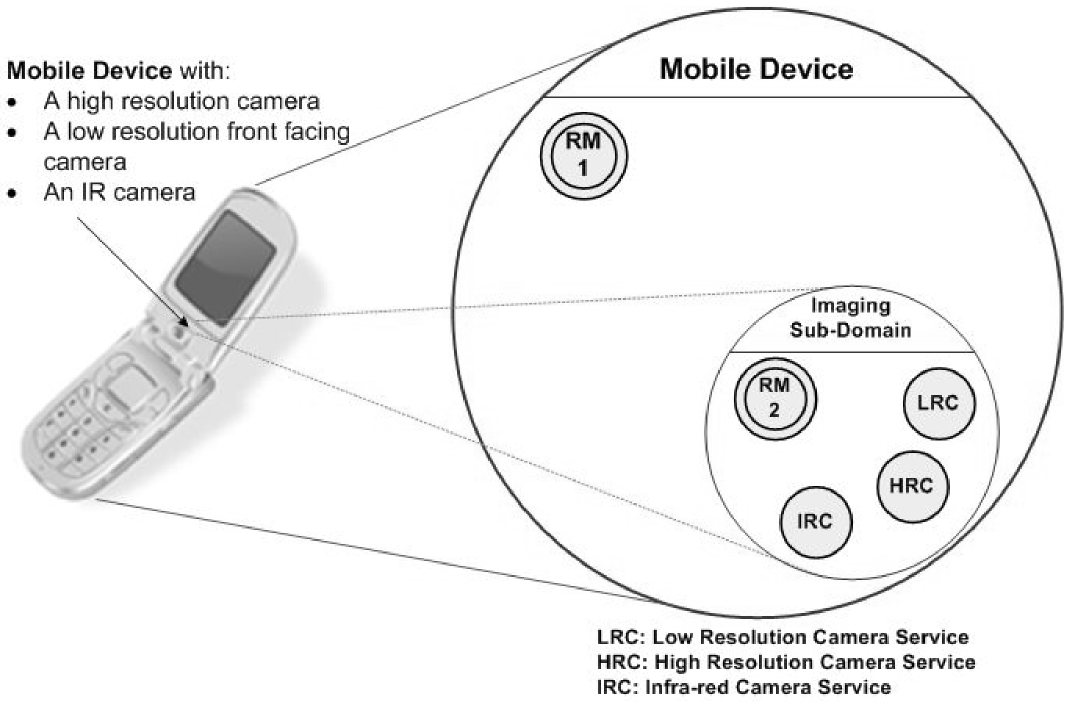

Mobile Device with:
• A high resolution camera
• A low resolution front facing camera
• An IR camera
Mobile Device
RM 1
Imaging Sub-Domain
RM 2
LRC
HRC
IRC
LRC: Low Resolution Camera Service
HRC: High Resolution Camera Service
IRC: Infra-red Camera Service

FIGURE 4.6: A Mobile Device with Imaging Services as a Sub-Domain

the Service implementation standards within the meeting domains, additional Services might be needed to facilitate the actual exchange of information. If the Domains follow different protocols and naming conventions, for example, then additional Services like a Translation Service are necessary to interoperate. This Service is transparent if the Domains implement the same standards, implement the same protocols and use the same namespace, for instance. We discuss the Services required for network composition within domains in subsequent sections.

End to end communications across this architecture is non-trivial given the nature of network composition and domains (heterogeneity, isolation via boundaries etc). Certain domains (routers or ISPs) can offer specialized Services such as connectivity, packet forwarding, reachability to other domains or even a generic Service which provides ‘connectivity to global Internet’. Other domains (gateways) can offer connectivity to a very specific set of Services (following policies under a restricted administrative control). These domains can impose certain restrictions to other domains wanting to use such Services. Domains can also specify and maintain a ‘Domain Ontology’ to classify

Services. This can improve Service capability descriptions, dependencies among Services, Service Discovery and matchmaking, but will need a translation Service to enable interoperation with other domains following different ontologies.

RM Discovery and Negotiation

The primary step involved to setup a relation between two communicating domains is to agree upon the standards, data formats, policies and other conditions. This requires negotiation between the RMs of the respective domains. The collection of communicating domains with RMs form systems composed of multiple autonomous agents and negotiation enables them to arrive at a mutual agreement regarding their relationships [169]. Since, this process creates the connecting environment for the domains, it is necessary that a rich semantic language be available to them to negotiate [170]. However, there is no single language that forms a standard negotiation language. Besides, the assumption that another RM (thus the domain) speaks a specific negotiation language is an unsafe assumption. We propose the use of a *meta*-negotiation language [25, 171]. This initial exchange (Fig. 4.7) concludes with the negotiation language supported by both domains, the semantics of negotiation, policies applicable to the negotiation and other related aspects. Once the negotiation language has been agreed upon, the domains can exchange capabilities, requirements and policies to setup a relation between them. Once a rich negotiation language has been finalized, the Domains can exchange information to establish a 'relationship'. This relationship forms the basis for all the further communication among the Services of the domains.

4.1.2 Visualizing Relationships

We define 'Relation' as an association among dynamically collaborating nodes, devices and services in a network, characterized by a 'relationship metrics'. We propose a frame work termed 'Relationship Oriented Service Architecture' (ROSA)[172] to agree upon a

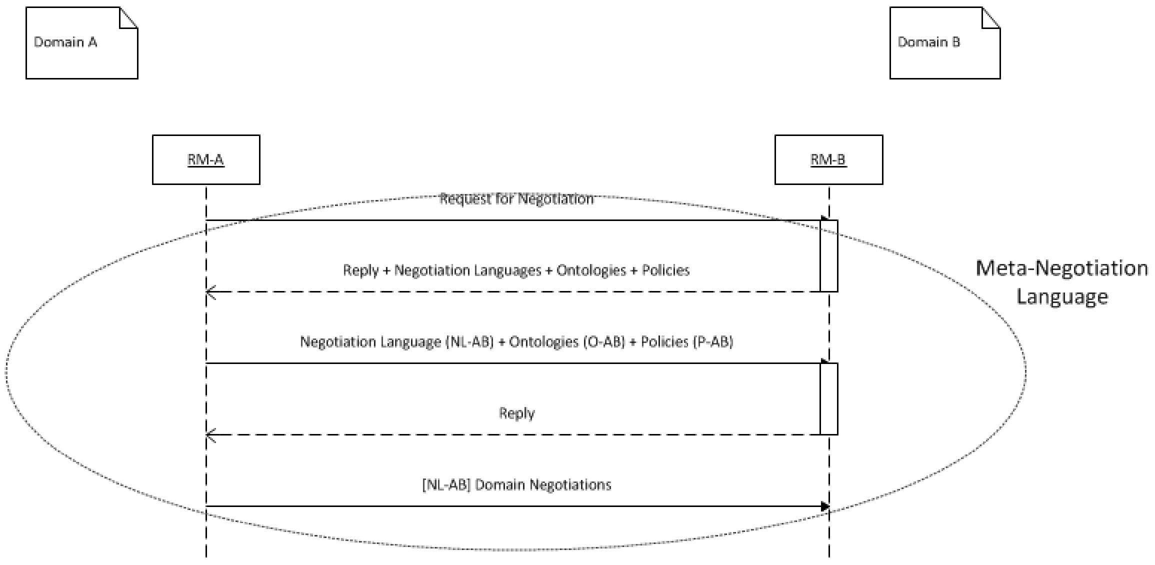

Domain A
Domain B
RM-A
RM-B
Request for Negotiation
Reply + Negotiation Languages + Ontologies + Policies
Negotiation Language (NL-AB) + Ontologies (O-AB) + Policies (P-AB)
Reply
[NL-AB] Domain Negotiations
Meta-Negotiation Language

FIGURE 4.7: Domain RMs agree on Negotiation Language, ontologies and policies using a *meta*-negotiation language

broad vocabulary that will be used to model recurring themes in communication domain and integration environments. The aim of such a framework is to be able to reference one or more open specifications or standards for each identified service, that can be used to implement various version of the service. This is a flexibility available in the SOA. SOA principles are considered in all aspects of design including interfaces between modules and the relationship description. This enables us to decouple certain aspects or modules and develop it independently. Besides, the services provided by some of the modules can be accomplished by a web based service or remote entity. This would be helpful in abstracting the network architecture across domains and networks without too much reworking of the communication network. This should also simplify integration with existing infrastructure and third party service providers (like standards based trust and security service providers).

The relationships in the first generation of ROSA architecture is restricted to a few well defined associations due to technological limitations and to reduce complexity. We define four types of relationships among domains for ROSA:

Resource Sharing Relationship In this relationship, domains keep control over its own

Services, and policies are applied within domains. RM acts as a gateway Service for communication between domains and exerts control over requests and responses traversing through it. Example scenarios are domain roaming between two operators, One domain using a connectivity Service of another domain to access external resources.

Control Sharing Relationship Domain boundaries remain the same, but RCS can share control over some of the Services from peering domains. This shared control means that both domains can advertise these shared Services as available within their own boundaries. Example scenario could be a business meeting where the control of a shared whiteboard is dispersed among the participating devices to be used.

Control Delegation Relationship Control of certain Services within a domain is delegated to another domain. This is an extension of the Control Sharing Relationship where the Service disappears from the original domain and is available only at the delegated domain. Example scenario could be using a authentication Service of a visiting domain.

Domain Merging Relationship A relationship when two domain negotiate and merge into a single domain (as presented to other external domains), with a union of Services from both the domains. Example scenarios could be a PAN becoming a part of the home network or few domains merging to become a larger domain (during business merger).

After negotiations (Fig. 4.7), Domain A will have a clear understanding with Domain B (and vice-versa). The information outcome of negotiations include standards, data formats, contexts, domains policies, Services available, Service descriptions and associated policies, QoS etc. Since we associate and extend the 'Domains meeting' concept to all communications, it is usual for a domain to have numerous relationships with multiple domains at the same time. A 'Relationship Database Service' (RDS) is maintained by each domain to store and track relationships[1](Fig. 4.8). The RDS functions as a snapshot for the environment that the domains operates in. Domains can choose to be stateful or stateless in their relationship with a particular domain. For example, routing domains (traditional routers) can choose to be stateful of other routing domains only (to populate and maintain routing tables) while remaining stateless about the domains which just request a basic routing functionality. An extension to this is a partially stateful approach, when some state (preferably soft-state) information needs to be maintained to provide QoS for a specific flow. It is also possible, following the Service abstraction of

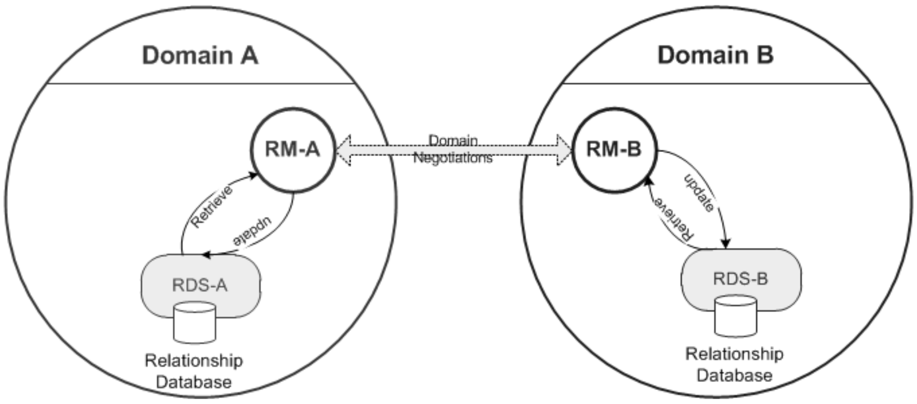


FIGURE 4.8: The domains use a Relationship Database to keep track of associations

domains, to virtualize another domain (with which a relationship has been setup) as a Service with its boundary. This virtual Service is specified within its Service Repository using the Relationship descriptions as interfaces, capabilities and policies. It is up to the domain to maintain a naming format to identify itself and uniquely identify another domain. It could be a global namespace such as IP, or a layered naming convention such as in FARA [41] or HIP identities or AN identities. The domain which identifies itself with a specific element of a namespace should associate enough information for collaborating domains to verify its identity.

The overall architecture provides an 'Intelligent Middleware' (similar to ESB) managing the communication, while providing an API towards the applications and managing connectivity resources independently (Fig. 4.1). In SOA, the actual services are usually

[1]The format and standards for the relationship entry and the relationship database need further study.

relatively simple ‘black boxes’ that can be applied in a flexible fashion in a variety of instances, with focus on minimizing duplication of functionality. While it is not efficient to dictate that all applications orchestrate and construct their own solutions from the a given set of services, any large scale application can do so via the APIs that these services expose individually. For normal scenarios, the separation of application logic from transport logic to make applications communication agnostic can achieved by providing a higher level aggregate functions of these services via more generic APIs. This is done via the ‘Relationship Manager’ (RM) [1]. We focus on the argument that the given modules can be composed into a coherent architecture via a single paradigm, namely *Relationships*. A relationship description contains parameters (‘relationship metrics’) to express the nature, context and background of the collaboration.

4.2 Service Oriented Analysis of Network Communication

To propose a network architecture within the structure of SOA, it is necessary to define a collection of services and define the environment (or a framework) within which they function. This is a non-trivial exercise, given that our intention is to define a generic architecture for the next generation communication networks. To arrive at such a architecture, we inspect the highest level abstraction required for any communication. A service oriented analysis of communication architecture need to be done to compartmentalize capabilities and functionalities into clearly defined functional boundaries. These can be later assigned service boundaries, keeping with the design principles discussed in section 3.4. This is perhaps the most important phase of the architectural design. The architecture being specific to communication domains, other design principles need to be incorporated such as tussle, business borders etc.

[1]This concept is covered in subsection 4.3.1

Service Oriented analysis of network communication is necessary to identify features that need to be divided into services and iteratively analyze each service to identify capabilities. The traditional concept of networking is exchange of data between physically separate entities (traditional nodes). This implies that such approaches cannot be used to address physically co-located networks (in-device) with the same approach as the geographically separated ones. Our approach with the help of service abstractions facilitates a uniform approach to this problem. We partition the functionalities required with in communication networks into service boundaries using criteria described in subsection 3.3.2.

A representative set of abstract Services that can be used within our framework is presented in Figure 4.9. We regard this as a good set of Services for ROSA. However, thanks to the SOA approach, the set of Services is extensible and adaptive to a domain's needs. A brief description of the functionality and logic behind the Service boundary follows. The conceptual blueprint for all planned services is together indicated in the framework and is indicated as the Service inventory blueprint. The services that are in the framework are evolved via the *Service Modeling Process* which is a sub-process of the Service Oriented Analysis (section 3.4). The end result of the Service Modeling Process is the gradual creation of a collection of *Service Candidates*[1]. The Service Oriented Design uses a set of predefined service candidates from the service inventory blueprint from which actual physical service contracts are derived. The *Service Candidate* in the service oriented design represents a conceptual service within the inventory blueprint, whereas the *Service* indicates a physical implementation[2].

A selection of 'building block' services (service candidates), which can be orchestrated to form higher level services is depicted in Fig. 4.9 to visualize the framework. All these services candidates have certain set of capabilities, carried out using some solution logic. The descriptions of the capabilities that can be accessed by another service is expressed

[1]A Service Candidate is a conceptual service definition which represents a usable modular logic to solve a unique problem.

[2]Since, one definition essentially points to another, we use these terms interchangeably; differentiated only when it is necessary.

as its service contract. The name of services depend on the service models (task, entity or utility) and the process/function it will be automating[1].

We specify two kinds of mandatory services within this architecture. One kind of service *implements some logic internally* and the second *implements an interface to an external logic*. To the users of the services, this might be oblivious but is worth noting with regards to the services listed below. We partition the requirements in communication

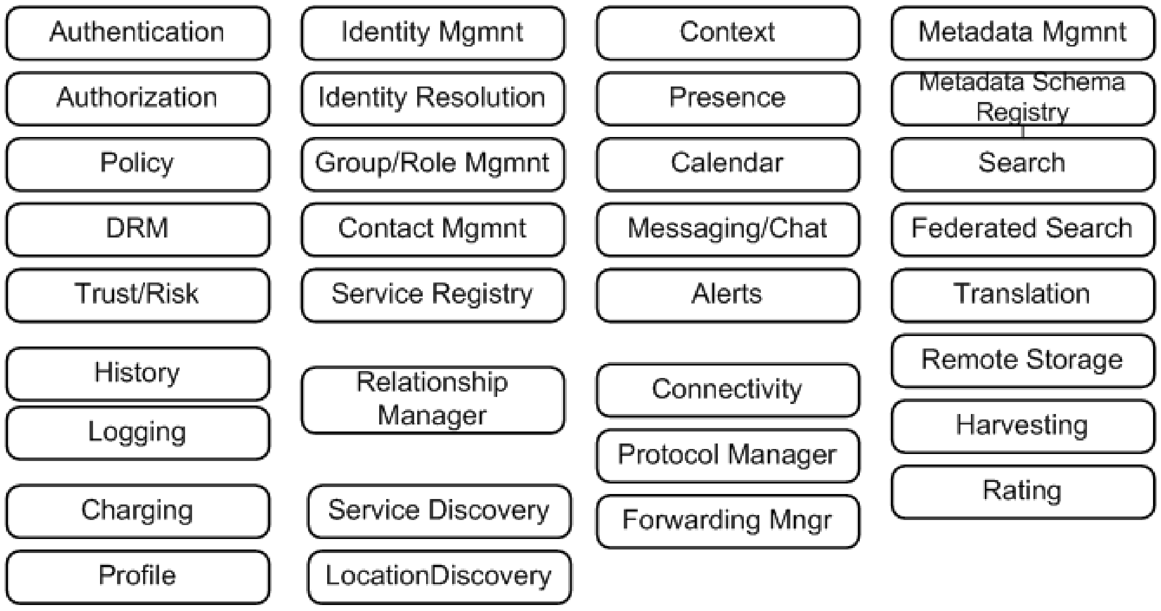


FIGURE 4.9: ROSA Service Inventory

domain into Services and present them as a list below. These Services form the Service framework for the new architecture. A collection of these functions (or a composition of the related Services) can satisfy the application requirements.

Alert: Dissemination of alerts, updates or announcements.

Authentication: establishes the identity of resources (nodes, services etc.).

Authorization: establishes the rights and permissions to use resources in a ceratin context.

[1]For better human readability, the operations for all services should be based on the format *verb* + *noun* and exclude the name of the service

Calendar/Scheduler: Provide/share a system wide calendaring and scheduler for user/applications/other services.

Charging: Manages charging related functions for applications.

Connectivity: Provides information regarding the status of connectivity resources available with rich parameters like speed, cost, reliability, trust etc. Abstracts interfaces available for communication.

Contact Management: To manage a system wide repository of contacts (other users) with rich metadata.

Context: Maintains information about the nature of the activity a user is currently engaging in, with relation to information such as role, location, presence/status etc. Also supports registering and deregistering of context by applications.

DRM: Supports the allocation and application of rights policies against resources through standards (e.g. REL) to determine access.

Federated Search: Supports the processing of search across multiple repository types, such as a combined search using SRW, XQuery and Z39.50 protocols against repositories maintaining a range of different metadata formats. Aggregation and a unified presentation format can be supported.

Forwarding Manager: supports different forwarding algorithms to provide suitability of next hop for transmission.

Group/Role management: To manage group/community information, group memberships, credentials and roles for access management.

Harvesting: Supports harvesting copies of some or all metadata and/or resources.

History: Long term preservation and managed destruction of metadata about connectivity, data, transactions etc.

Identity Management: manages the creation, registering, deregistering and persistency of identifiers for resources such as local labels or HITs [51].

Identity Resolution: Use identifiers and other metadata to provide current location or redirection locations for resources. Also supports cross-mapping of values in different identity namespaces.

Location Discovery: supports the proximity awareness of a node, by providing support for own location, neighbourhood discovery and related information.

Logging: Provides generic logging services for applications/debugging.

Messaging/Chat: Support for one-to-one or multi-entity messaging and chat management. also supports broadcast of messages to users or groups.

Metadata management: Supports the management of metadata for resources.

Metadata Schema registry: Supports the registration of metadata schemas and retrieving definitions of entities.

Policy: supports access to, creation and management of rules and policies, to facilitate access management or activity processing. This Service maintains business constraints within and across domain boundaries.

Presence: Provides information about the status of a resource or user.

Profile: Supports maintenance of machine readable information on user or application preferences.

Protocol Manager: Encapsulates available protocols for late-binding and makes available a list for late binding.

Remote Storage: Supports access to remote/decentralized storage facilities like ‘Cloud’, SAN etc.

Reputation: Support creation, management and use of secondary metadata such as user ratings and annotations. This Service can use data created by the Trust/Risk Service as an input and provide ratings of Communication Services to other Services.

Search: Fast search implementation for a single query grammar over a single repository (mostly local).

Service Discovery: Encapsulates various service discovery protocols to be used by applications or relationship manager. The *Service Discovery* module can query over metadata that describes services.

Service Registry: Supports maintenance of a repository of available services.

Translation: Supports transformation of information between incompatible formats.

Trust/Risk: Manages Trust/risk scores associated with resources including creation, updating, and evaluation of such scores

The core task of creating the above described framework is to identify a broad set of services that need to be defined (called 'factoring services'). We consider factoring to be an ongoing process as experience informs the choice of services, identifies shortcomings and indicate the services that require creation, discount, splitting or joining. We propose Services here as a definition of function and scope. The functional focus provides the capacity to be specific about the range of expected behaviour of individual service, while being agnostic with regards to the implementation details or design of solutions. While this definition is far from sufficient to implement a network architecture, it provides as a starting grid to more detailed specifications and a reference model. From the abstract models of services, it is possible to derive XML schemas that define data to be exchanged. This approach enables specific SOA standard based definition for services to be implemented (as a webservice, for example). Such a framework is realized in an application with an interface to access a service that has commonly agreed operation definitions (e.g. WSDL) and data structures (e.g XML schemas) [98].

Based on the above patterns and Web-Services based frameworks like WebTransact [173], we derive the following conclusions. Any inconsistencies arising from the differences in competing protocols or data types (semantic dissimilarities), a *Manager* Service to abstract them should be the norm. These Services (which together form a *Mediation layer* as in WebTransact) provide a homogenizing layer between the Web-Services and the compositions. Semantically equivalent Web-Services can be aggregated as a new abstract Service exposing a homogenized interface of those Web-Services, facilitating Service compositions on top of the abstract Service. For example, to conceal differences or inconsistencies in data types, security policy or context of Web-Services, a communication manager is required to translate messages between different transport protocols like HTTP or SMTP. A security manager Service can traverse firewalls and handle authentication and authorization. A content manager can manage conversions between different document representations or data types. QoS is an area of varied Tussle. A composite Service (QoS manager) to monitor QoS metrics of component WS is necessary to delegate and manage sub-Services dealing with specific parameters and varying evaluation logic. These abstraction form the Logical reference Architecture for the proposed Architecture and we call it Relationship Oriented Service Architecture (ROSA)[172].

4.3 The Relationship Oriented Service Architecture

We propose a logical framework termed ‘Relationship Oriented Service Architecture’ (ROSA) to agree upon a broad vocabulary that will be used to model recurring themes in ICT and integration environments [174]. The aim of such a framework is to be able to reference one or more open specifications or standards for each identified service, that can be used to implement various versions of the service. This is a flexibility available in the Service Oriented Approach (SOA). SOA principles are considered in all aspects of design including interfaces between modules and the relationship description. This enables us to decouple certain aspects or modules and develop it independently. Besides, this also implies that the services provided by some of the modules can be accomplished by a web based service or remote entity. This would be helpful in abstracting the network architecture across domains and networks without too much reworking of the communication network. This should also simplify integration with existing infrastructure and third party service providers (like standard based trust and security service providers).

We can observe the scalability and flexibility of our architecture in the logical reference architecture. Different logical workflows can be composed to form different logical architectures and can later be implemented to form reference models and ultimately working solutions. The smallest common denominator Service is the Relationship Manager since it handles the basis for the architecture compositions, i.e, Relationships. Additional Services contribute the necessary functionalities desired from the architecture. We cover two such reference architectures in this section. One is a simple Network Architecture which can address basic networking functions, using pre-existing Services to provide matching queries. The second instance of the architecture provides a much more complex snapshot of a network Architecture where Solutions to queries which do not have pre-configured solution Services are composed on the fly out of existing available component Services. The latter case requires more advances in technologies than that is currently available.

As discussed previously, Services can be differentiated based on their functionality, scope and dependencies. The most useful basic Services addresses a single task and is very reusable. However, the functionality attached to these services are minimal (mostly, a single purpose). A solution to common network tasks require many such Services working together as components to form a much more complete albeit complex composed Service. Such composed Services can be used by Applications to implement their solution logic. The interfaces to all types of Services remain the same, but the composed Services provide a much more usable Service to the applications (in our case, via the RM). From the concept of a domain of Services, as covered in Figure 4.3, when two domains meet, the RM forms the well known point of initial contact. To share the domain Services with another domain (or in our architecture, to establish a *relationship*), a significant quantity if information needs to be exchanged. This includes, but not limited to, list of capabilities and the list of Services available within the domain boundaries, parameters and additional information attached to these Services, policies and other usage related criteria and so on. If a domain (or a Service within that domain) requests a functionality from another domain, then a user/provider relationship needs to be setup between the domains. This involves exchange of queries, replies, policies and other negotiations. We delegate the responsibility of negotiation handling to a dedicated Service named *Negotiation Manager*[1]. We will inspect the essential Services required to compose the simple network architecture below.

4.3.1 Relationship Manager

The relationship manager (Fig. 4.4) is itself a service which orchestrates other available services and makes available APIs to accomplish most of the common services that applications need. The RM fits the numerous service components into a logical process (Orchestration) and facilitates the translation of data flow between services that may interpret a term differently. RM helps to avoid the 'monolithic silos' traditionally formed when implementing a complete usable service. The RM orchestrates the services for processes (on behalf of applications), based on contexts. Each entity communication space must provide a set of objects, the services of which an entity, process or individual, can use to achieve his goals. Entities always communicate with objects in their environment according to a certain context[2].

By being a service by itself, the RM is also replaceable. However, to be truly modular and avoid tussle in the core, the relationship manager should be minimal, analogous to a

[1]See section 4.4.1 for more details.

[2]A context represents a 'universe of discourse' in an entity communication space. It defines relationships and causalities of an entity to and between particular objects of the relevant communication spaces [175]. The context is expressed in the context service and the RM contains the logic to use it for orchestration.

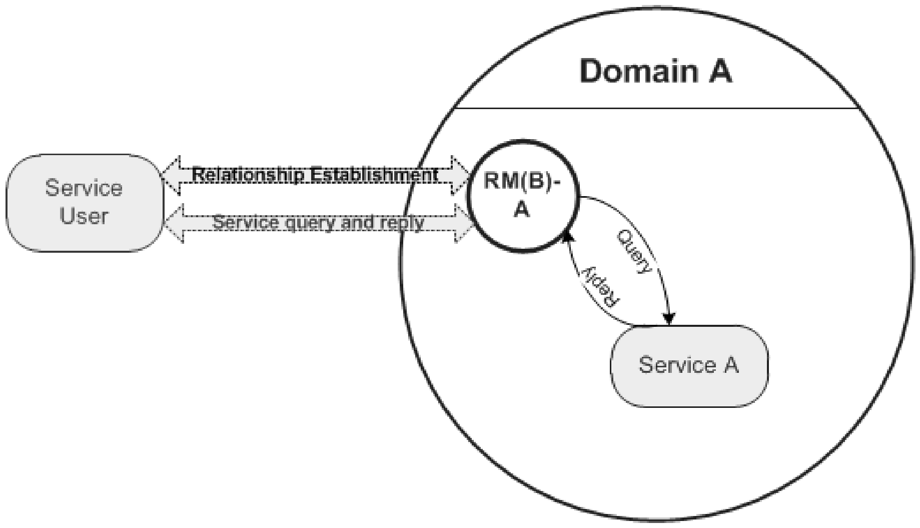
micro-kernel in an operating system. This means that all useful services should be implemented outside the RM, or in services space and the only service that the RM provides is a meaningful orchestration. However, even for environments where the applications directly manage the orchestration, RM must be present for bootstrapping purposes.

From a different perspective, the RM virtualizes the available services to the applications. This is similar to the concept in Fig. 2.4(b) where the Service Bus virtualizes the candidate services from the requestor's point of view. The RM can be thought of as a local instantiation of a Service Bus, which simplifies the search, query, binding and access of other services within the ROSA framework.

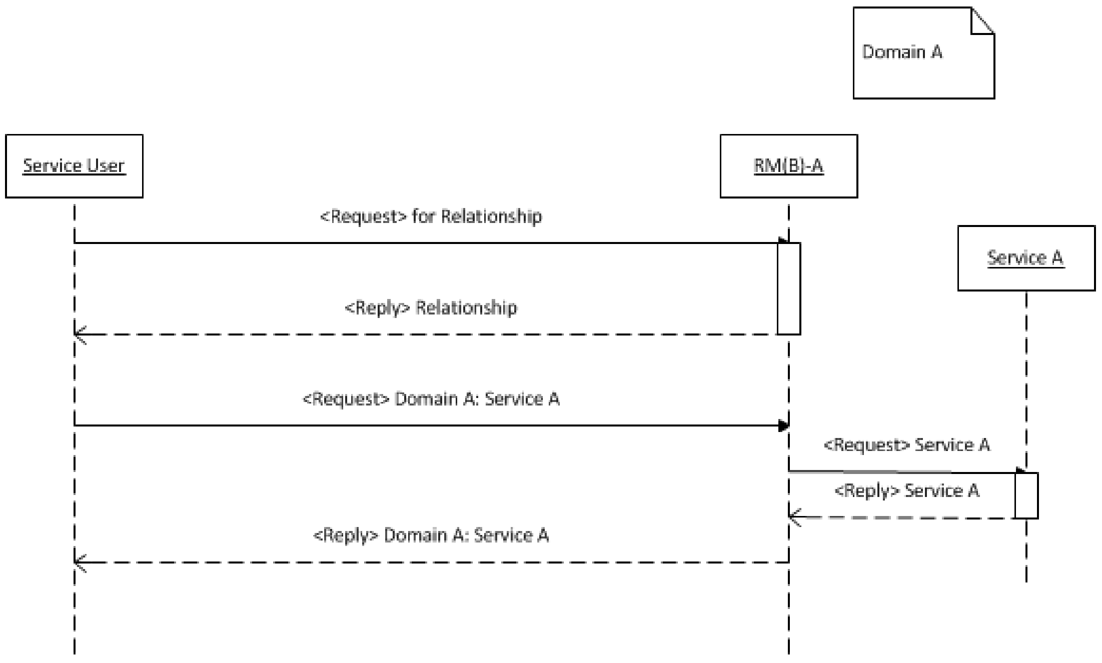
RM can operate in two modes, the Broker Mode (RM(B)) and the transparent mode (RM(T)). These modes also influences the architecture of the composed network. In RM(B), all external requests to the domain are routed, interpreted and handled via the RM exclusively (Fig. 4.10). This is the simplest option for an external application/service and requires the most intelligence from the RM. However, this mode can be inefficient for many large scale applications that want to use individual Services available within a domain and have enough intelligence to compose them. The latter can request the RM(T) mode (Fig. 4.11), where the RM is in observational state and do not intercept the service requests from external domain (a certain amount of monitoring is still warranted to ensure security and enforce domain-wide policies). RM is the only Service which maintains and overview of all resources in a domain and is able to apply policies in using them[1].

The requesting Service (User) initiates a request for relationship, by querying a well know Service (here, the RM). The RM replies to the query with more information on its own capabilities and Domain wide mandatory profiles (applicable to itself as well). This step initiates further negotiations between the User Service and the RM.

[1]SIP offers similar flexibility in using Services

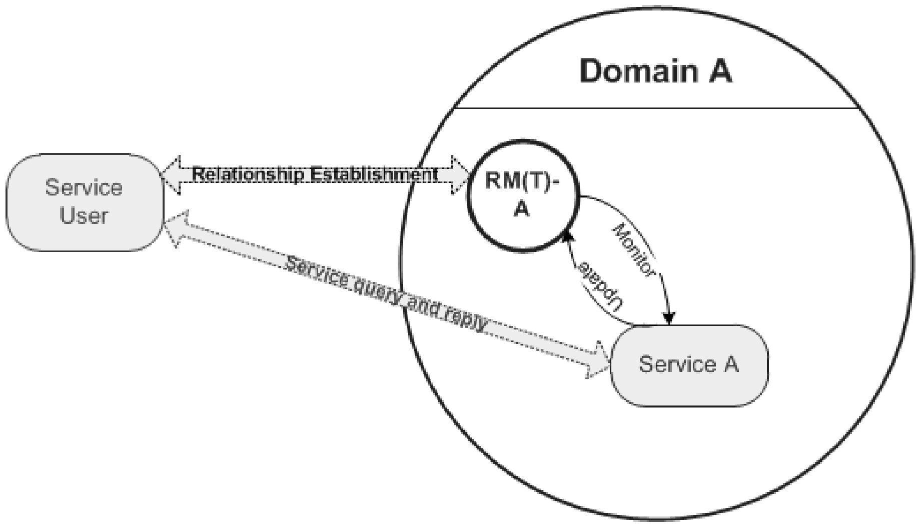


(a) RM in Broker Mode (RM(B)

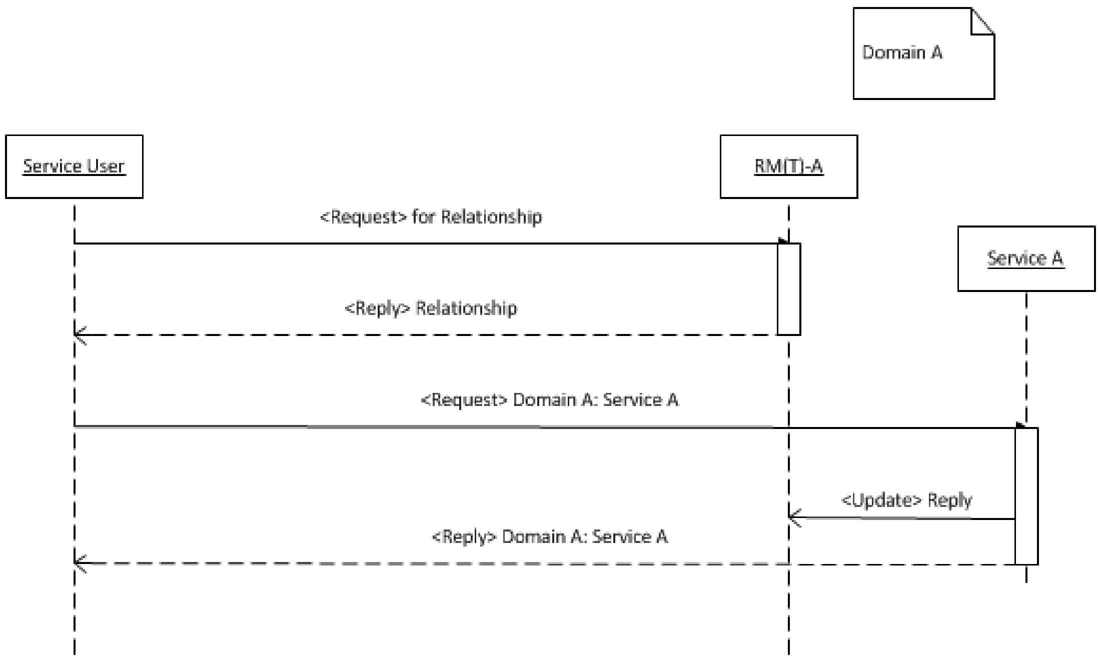


(b) Message Sequence for RM(B)

FIGURE 4.10: RM in Broker Mode - RM(B)



(a) RM in Transparent Mode (RM(T)


Domain A
Service User
RM(T)-A
Service A
<Request> for Relationship
<Reply> Relationship
<Request> Domain A: Service A
<Update> Reply
<Reply> Domain A: Service A

(b) Message Sequence for RM(T)

FIGURE 4.11: RM in Transparent Mode - RM(T)

4.4 Instances of ROSA

In this section, we demonstrate how a network architecture can be composed out of the framework components discussed till now. Given the numerous choices that can be made for Services and their combinations, there could be any number of network architecture instances that can exist. However, the basic approach of deriving the architecture from the framework remains the same. The main steps that need to be done is the component Services within a domain, logical structuring of the Services around RM and the relationships the domain can hold with other domains. Once these requirements have been formalized, the following models are specified to clarify how the domain handles Service requests from Service Users:

- Component model
- Orchestration model
- Data model
- Transaction and error handling
- Quality of Service model

These models specify how the Services fit together, how they are invoked (sequence, for example), the data formats used and how to manage and recover from errors and exceptions, among other things. These can be specified using syntax based standard languages (for example, BPEL) or semantic languages (for example, DAML-S) or even a proprietary language[1] and are domain dependent.

The ROSA architecture is flexible enough to accommodate domains with varying Service profile. A router domain's priority is to forward information based on a lookup table (traditional router functionality). An extended router with QoS guarantees (or even DPI capabilities) can have more Services incorporated than just lookup and forward. A

[1]This practice is not recommended since it leads to inefficiencies and maintenance issues.

mobile phone can have a variety of other Services, but can also incorporate the routing functionality (to act as an intermediate router in mobile ad-hoc networks, for instance) by implementing corresponding routing Services.

We present the first steps in composing two instances of ROSA, the first generation ROSA and the next generation ROSA. The first generation ROSA is proposed with current technological limitations in perspective.

4.4.1 ROSA (Mandatory) Control Services

Some of the control Services in ROSA are mandatory since they are necessary to perform certain recurring functionality. We list the mandatory ROSA Domain Control Services (DCS) below:

Negotiation Manager (NM)

To establish a Relationship and further to enable the Service User to find services which best fit to its requirements (functional and non-functional properties), service users should negotiate and communicate with RM (in RM-B mode) or RM and numerous available services (in RM-T mode). Non-functional requirements of a service execution are usually QoS parameters, and are expressed and negotiated by means of Service Level Agreements (SLAs). SLA templates represent empty SLA documents with all required elements like parties, SLA parameters, metrics and objectives, but without QoS values. Since it is possible that the communicating Services might not have prior knowledge about the negotiation protocols before entering the negotiation or that they have matching SLA templates. This is specifically true for environments where services are discovered dynamically and on demand. Hence, *meta-negotiations* are required to allow two parties to reach an agreement on what specific negotiation protocols, security standards, and documents to use before starting the actual negotiation [171]. The Negotiation

Manager assists the RM with meta-negotiations, SLA mappings and if necessary, actual negotiations before the Service is actually invoked and used [1].

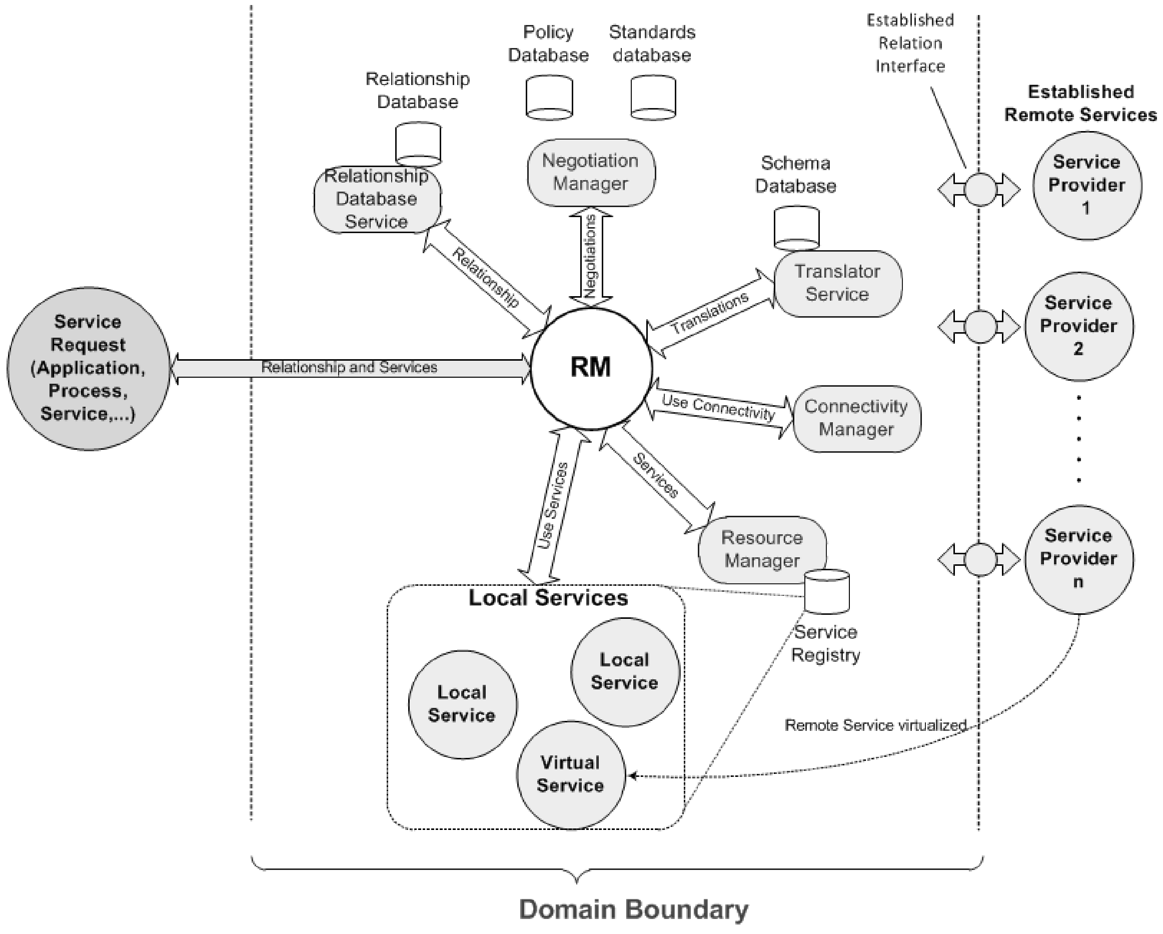


FIGURE 4.12: A first generation ROSA instance (Domain Control Services)

Relationship Database Service (RDS)

This Service interfaces the relationship database to other Services. The relationship database is populated with the established domain relationships with other domains. This Service also enables policy management since domains can define relationships for other specific domains (such as interactions rules and security requirements) prior to negotiations. The format and model for relationship representation needs further research and standardization.

[1]More on various implementation details can be found in [25]

Resource/Service Manager (RS-M)

The default service registry standard used in Web-Services is UDDI . But, as service capabilities continue to increase, the metadata along with the discovery and description mechanisms are susceptible to change. To accommodate for this evolution, we interface the service registry via a Service Registry Interface Service, whose purpose is to provide a uniform interface to heterogenous service registries or databases available within the domain.

The Resource Manager could directly provide generic abstraction for various network technologies used and the network infrastructure. Or these functionality could be separated and managed by a Connectivity Service that interacts with the underlying connectivity resources and provide a generic interface (Fig. 4.12). Other Services can utilize the connectivity resources though the technology independent methods provided by the Connectivity Service. Further, applications can use the RCS to establish, maintain and terminate end-to-end connection across domains.

Translator Service (TS)

The Translator Service (TS) converts between various representation formats in order to aid flexibility and aid modifiability for Services and standards. This service transforms/translates between the external language used by the participants and the internal language used by other services such as the composition generator. Each step/service might require different protocols and procedures.

This Service which provides request and language translation facilities distinguishing and independent evolution of external and internal service specification languages. Service users can specify requirements in a relatively easy (descriptive) manner and increase accessibility. Internal languages (for instance, input to the Composition Generator in next generation ROSA) need to be more formal and precise for automatic processing.

An example of this operation can be transforming input from users in WSDL or DAML-S to a language for composition generation.

4.4.2 ROSA (Optional) Control Services

These Domain Control Services described above follow the SOA paradigm and are distributed and modular. While the mandatory DCSes are required to make the ROSA flexible, more optional Services could be incorporated to vastly improve various factors of the architecture like efficiency, security, reliability etc. Additional domain specific Services add completeness to the architecture. Some optional Domain Control Services are mentioned below:

Service QoS Certifier

This is an optional Service, which significantly improves the reliability of the architecture. Every Service follows a life-cycle of its own, as an independent entity. The functionality (or the Quality)of a Service can vary over time or context. It is essential to have an arbitrator which can clarify the available functionality, the accuracy of Service claims and Other QoS claims put forth by the Service provider. We propose a QoS Certifier (Broker) Service (ROSA-QC) to handle this functionality. The parameters verified by this Service includes performance and dependability related attributes, especially non-functional attributes like cost, payment. A provider and Service independent approach for evaluation of QoS attributes of Web-Services is covered in [176]. Apart from verifying QoS claims, this Service can enhance trust and reliability of compositions and Services available by incorporating reputation as a parameter within its considerations. Since, ROSA-QC Service is a Broker Service, it can utilize third party reputation and trust providers as delegates. Reputation is a subjective assessment of a characteristic or an attribute assigned to one entity by another based on observations or past experiences [177]. The ROSA-QC Service relies on claimed provider qualities and past interaction experience to establish trust among known and unknown participants. Even the reputation of newly created or known Web-Services can be evaluated using various techniques [178].

Composition Generator

In the real world, it is possible that every query cannot be fulfilled by an exact matching Service. However, if the query is broken down into a series of simpler requests, it might be possible to satisfy each with appropriate Services. This implies decomposition of the request into sub-requests to find appropriate matching solutions. Another option is to build a composite Service out of the Services available within the domain to provide a solution for the request. This means that Service composition can be part of the solution process. For a flexible and efficient network architecture, automatic on-the-fly composition generation is necessary to produce meaningful composite Services out of what could be possibly numerous available Services. This function is handled by the Composition Generator (CG). For each query, the CG tries to generate a plan that composes the available services in the service repository to fulfill the request. [179] discusses a method for dynamic selection of optimal service from several semantically equivalent Web-Services based on performance criteria. Similar work is also carried out by Oldham *et al* in [180], but where matching is based on SLA semantics.

There are numerous approaches to create compositions out of component Services. One is eflow (graphs). Another approach uses Polymorphic Process Models (PPM) where composition is modeled as state machines and transitions; where dynamic composition is enabled by reasoning based on state machines[181]. Another approach using AI planning (Situation calculus, PDDL[1], rule based planning, theorem proving etc[181].). The specific method by which the compositions are achieved is beyond the scope of this work.

[1]PDDL is a widely recognized and standardized input for planners.

Composition Evaluation/Verfication

This service is essential to implement an automated and dynamic composition method within the Composition Generator. The Composition Generator can produce more than one composite Service fulfilling the request or requirement. This is possible since many Service present in the repository can have similar functionality. To select the most appropriate composite Service, the different options are evaluated based on non-functional requirements and attributes such as QoS and Security. Without delving into implementation details, there are many methods that can be utilized such as assigning various weights to various attributes and ranking the compositions. Another approach is to express the QoS requirements in an appropriate language and compare the compositions for match or compatibility.

The composite service (or services) proposed by the composition generator needs to be checked for its correctness with regards to constraints and context, and to verify the validity of the proposed solution. While this could be achieved within the composition generator itself, it is safe to abstract as a new service since model checking and verification methods are dynamic and could be provided independently of the composition approach, model or methods. This service can will require the composite service conditions (input, output and states), user and component service requirements (input, output), domains policies and parameters derived from information such as profiles and contexts, to name a few. Evaluation of semantic based service composition [182]. In a next generation ROSA domain, the compositions are generated automatically. It is ideal to verify the correctness of such compositions before executing them. However, the current state of verification technology is not advanced enough for automated verifications. Besides, the tools ecisting today are insufficient to automatically translate to a verification notation (or languages such as PROMELA, for example) and verify it. The validation of a composition of complex Services is very likely to have too large a state space to be able to be verified in a reasonable amount of time. A more convenient approach may emerge in the future for automatic runtime verification of dynamic compositions, wherein the functionality can be included as a Service within the framework. Till then (an in first generation ROSA) this is more of a token Service to implement a rudimentary verification such as checking process flow, input, output for inconsistencies and adherence to domain policies.

Execution of a Composite Service

Once a unique composite Service has been selected, it needs to be executed. It can be visualized as a sequence of message passing according to the composition model. Data flow can be modeled as the output data of a former executed Service transferring to the input of the next atomic Service to be executed. Result dissemination and cleanup after execution including listing successfully completed composite Service as a Service candidate in the registry for future use.

The RM as a coordinating Service must oversee and manage contingencies in case of errors or exceptions during normal functioning. This is specially relevant for pervasive, mobile and highly dynamic environments. It has to determine the alternative course during unavailability or replacement of a Service within a given composition. RM can delegate exception management to other Service, like the composition generator to be reinvoked to find a replacement Service.

4.4.3 The first generation ROSA

The first generation ROSA suggests a reference Architecture for a simple Network Architecture. We assume in this instance that the requests that are encountered by domains involve previously known Services and no intelligence apart from selection of the appropriate solution Service is necessary further. This limits the usability of this architecture, but is necessary due to the technological limitations in the early introduction phase of such an architecture.

As we have discussed earlier, negotiations form the core of establishing a relationship between domains and are handled by a *Negotiation Manager*. The relationships formed are codified and stored in a relationship database and is accessed via the *Relationship database Service* (RDS). These form the core Services of ROSA, along with the RM to control the network composition. We call these 'Domain Control Services' (DCS) and are minimum requirements for ROSA. Apart these, there are a few other Services that is required for a simple network architecture. One important requirement is a repository of locally available Service descriptions, their capabilities and related parameters (including QoS related criteria). We introduce a broker Service called Resource/Service Manager to address this requirement. A visual representation of this architecture is indicated in Fig. 4.12. The proposed mandatory Services (DCSes) within this Reference Architecture are listed below:

As mentioned earlier, the Translator Service provides certain flexibility for Service users to specify their requirements. Depending on the expressiveness or formalization of the requests, the domain ROSA architecture composition complexity varies. For instance, if the request is formulated as 'an abstract composition/process model + set of tasks + data dependencies + additional QoS parameters (or other clauses to select atomic Services)', then the domain ROSA only needs to select the appropriate Services available within the domain registry to fit the model. On the other hand, if the request is a 'task + constraints + policies', then a dynamic composition might be required.

4.4.4 A next generation ROSA

The approach of simple ROSA holds true as long as request from Service users matches the functionality and criteria provided by existing Pre-defined Services. This approach is not flexible enough to incorporate an anomalous or unprecedented Service request, the solution to which might be the combined functionality of several Services (the sequence of which is not pre-defined). This is a restriction to the dynamic nature required for the architecture and automatic Service Composition is vital for this functionality. The

Service composition of Web-Services by itself is a subject of intense research, and it is beyond the scope of this work to specify which methods or technologies to adopt. But, it is essential to provide place holders for the *Tussle* to play out in this context. We provide an example reference architecture for such a scenario where, the architecture exploits the ability to automatically compose itself from component Services.

The distributed versus centralized approach for DCSes are considered in the design of the first generation as well as the future generation ROSA. A centralized control logic and resource management (a monolithic Service) would be the ideal approach for performance. But, distributed approach of control and resource management have their own advantages such as separation of concerns to ensure security and privacy, extensibility, flexibility (both business and operational) etc. We weight these options and implement a partially-distributed system architecture where the separation of concerns are achieved via distribution to Service managers but avoid the expensive and complex choreography among them to attain a network composition. Instead, these Services provided are orchestrated by a centralized function (RM) to make decision. This approach becomes significant in the next generation ROSA when Services are handled automatically and network composition is dynamically generated and used. With the numerous number of Services available within a domain, an ad-hoc or distributed composition strategy without a central controller might lead to scalability and complexity issues.

The reliance on RM to aggregate and manage the domain specific functions demands high reliability and availability from its implementation. In our architecture, ROSA-RM plays a core role in setting up and maintaining the work flow of the composed network instances. When various application or processes are concurrently using the network connectivity through ROSA, the RM will be handling multitude of connections with state data and various other information associated with it. The scalability of the architecture will be crippled if RM has to maintain all this data concurrently by itself. We propose to achieve high resilience for the RM using a state database interfaced through the RM state Manager (ROSA-RMSM) using full architectural state management deferral or internally deferred state management [5][1]. The ROSA-RM can push various state and transaction data relevant to processes to the database when not needed (within the life time of that transaction) and remain active with minimal overhead. The state database is also helpful in restoring the architecture from exceptions where RM needs to restart and *remember* the states of running transactions without having to terminate them. Thus, the ROSA-RMSM provides reliability, redundancy and scalability to the architectural core.

For a next generation ROSA, we propose the following Domain Control Services (DCS) to delineate the required functionality into boundaries.

4.5 Domain and Network Composition

In our architecture, we treat all Services in a uniform manner. All follow the SOA paradigm with interfaces and capabilities defined using Web-Services standards. In ROSA, there are two different compositions required, based on the candidates for composition. One is the ROSA domain composition, wherein the Services within a domain boundary are combined to provide a composite functionality.

4.5.1 ROSA Domain Composition

This is a localized composition in the sense that this composition occur within the boundary of a domain and the process is governed by the local domain policies and conditions. In the first generation ROSA, we expect these compositions to be manually specified and statically composed. In the next generation ROSA, these domain Services can be selected, composed and configured on the fly (dynamic and automated composition) to

[1]See section 11.4 *Measuring Service Statelessness* of [5] for a description on the techniques to achieve scalability and reliability via an external database

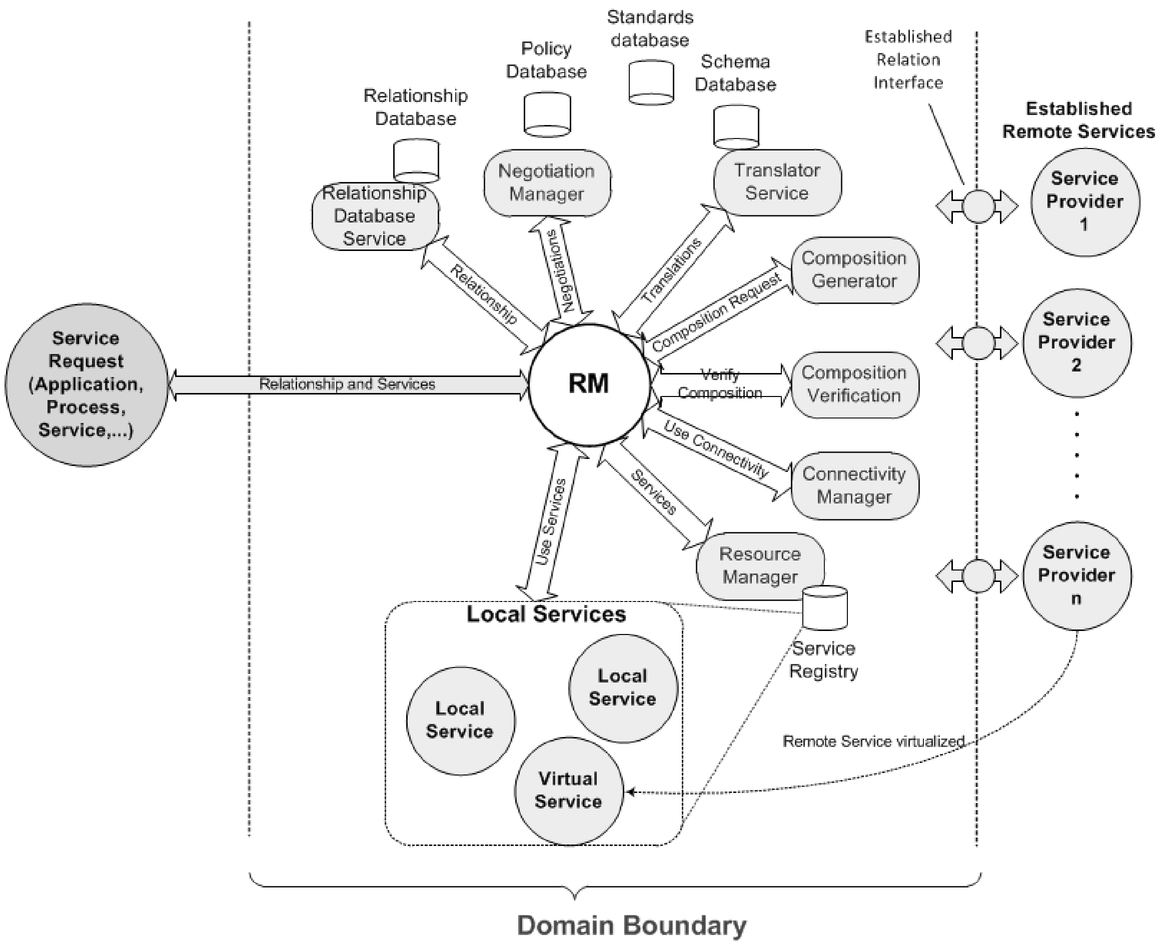


FIGURE 4.13: A next generation ROSA instance (Domains Control Services)

produce a composite Service. This enables the domain to respond to dynamic requests within and from outside the domain, with minimal human intervention. This design provides a uniform approach to provision locally available functionality in a Service Oriented approach, as Services. Legacy functionality could be wrapped in Web-Service specifications to be included as a ROSA Service and this take part in the next generation domain and subsequently network compositions. We inspect an example ROSA domain and a sample domain composition below.

The ROSA Domain Control Services (DCS) consist of a set of mandatory and optional Services that implement the domain and network control functionality. The RCS is a collection of Services and hence distributed, modular and extensible. The mandatory DCS have very familiar and identifiable symbolic names within a domain. For instance, RM must always be available at 'RM@domainA' (in an example domain 'domainA'). Similarly, the domain Service registry must be available at 'registry@domainA'. Similarly, each of the other DCS must be reachable at an agreed address 'name@domainA'. Any changes to this convention, must be updated in the registry of the domain. For example, if the 'negotiation@domainA' Service can be delegated to another Service (say 'negotiation@domainB'), it must be updated in the registry so that any Services looking for this Service can find a currently usable version. We assume that domain Services are capable of registering themselves (by a Service provider, a common domain discovery and registry Service) to registry@domainA.

The Service registration processes populate the Service registry with Service descriptions (using WSDL for descriptions, UDDI or DAML-S Service profile for repository). The properties to be specified in the registry include Service signature (inputs, outputs, exceptions), states of the Service (pre and post conditions) and non-functional values (QoS attributes like cost, Service quality, security, reliability). Each of the mandatory Services specified in the architecture have a specific part in the domain control function.

4.5.2 ROSA Network Composition

Where domains come together to form a network, the 'meeting of domains' is the abstraction for the network composition and is the basic action for network composition. Ideally, domains Services should be able to be composed based on requests from application in other peered domains as well. Network composition is based on relationship agreed by the domain controllers. Similarly to the AN concept of composition degree [183], relationships define the level of cooperation between the composing networks. Relationship can range from simple untrusted interworking where each domain exercise maximum control over its own resources that are being exposed to merging into a single domain (an extremely trustful scenario)[1].

[1]A new 3GPP standard [184] for network composition is evolving but is at a very early stage

Given the ROSA architecture, networks can be composed in phases. We identify five phases for network composition and life-cycle management listed below:

4.5.2.1 Network composition trigger

In this phase, a domain senses a change in context that calls for initiation of a new composition. This could be a discovery of a new medium becoming available, an internal Service request mandating a functionality not available within the domain (or already available relationships) or an application initiated request. The context for composition trigger can be changed based on the domain characteristics and policies such as security, energy conservation (for mobile devices) or just media sense (for ah-hoc networks). The outcome of this phase is the identification of a medium of communication and establishing a communication channel.

4.5.2.2 Domain discovery and advertisement

After composition trigger, domains pass into the domain advertisement and discovery phase, where depending on domain policies, it can:

- advertise (broadcast) itself with information including resources, capabilities, Services, standards etc to any other listening domains.
- passively discover by listening for other domain advertisements.
- actively discover by following a Service Discovery Protocol.

The outcome of this phase is that the domains to be composed are selected and identified.

4.5.2.3 ROSA domain negotiation setup

A basic interworking relationship is established through a meta-negotiation language. This phase includes agreeing on a common negotiation language, basic security, domain semantics and negotiation policies. The basic security could be authentication and authorization of participating domains either mutually or via a third party, generation and sharing of a cryptographic session key etc. The outcome of this phase is that the domains are able to negotiate in a rich expressive environment.

4.5.2.4 Relationship negotiation and establishment

Once the domains have established an environment for negotiations, they can commence the negotiations for a relationship agreement, which covers the following areas (not an exhaustive list):

- Nature of relationship
- Service capability utilization and policies
- Security and restrictions
- Semantics
- Exceptions and error handling
- Compensation, charging, billing, pricing etc.

All negotiations are carried out between the RMs (of the associated domains) with the help of Negotiation Manager. RM can further delegate parts of the negotiation among RCS to make the negotiations distributed in nature. After this phase, a relationship is established between the participating domains.

4.5.2.5 Relationship Agreement life-cycle management

Once the relationship has been established, the various participating Services and policies can be updated with relevant applicable parameters. There might be certain event triggers and other domain characteristics that need to set according to the relationship agreement and domain Service inventory to be updated.

4.6 Summary

This chapter specified in more detail a Service Oriented architecture along with the techniques and concepts for domain and network composition. The main control Services within ROSA and their functionality were introduced. The phases of ROSA network composition were also discussed which positions to overcome the static nature of the networks today. We formalized the relationships which form the basis for network composition and also inspected the ROSA control and domain Services.

We showed how Service Oriented domain builds up relations with other domains by using a ‘Relationship Manager’ and key support services inside a domain. We outlined a migration strategy from an early architecture with predefined sets of Services (first generation ROSA) to a future architecture with dynamic service composition and negotiation.

CHAPTER 5

OPEN ISSUES AND DISCUSSION

The driving factors behind this approach: context awareness, managing contingencies, heterogenous devices, empowering users/applications.

One of the significant questions that emerged when considering this topic was *'How will future networks look like?'*. There are whole industries constructed around this question with experts and non-experts venturing opinions[1]. It is hard to predict this future especially with pace of technology related evolution[186]. But, it is possible to infer a general direction in various disciplines by watching the emerging trends. The original Internet assumes interconnection of different sites possibly running independent applications and protocols. That concept was diluted with the proliferation of TCP/IP as the *de facto* protocol standard. This can be attributed to the single point of origin of standards and the unprecedented influence a single group (DARPA) had over the emerging networking community and other limited geopolitical and economic factors [187].

[1]See discussions about an *Internet Operating System* [185] implying a delivery technology/connectivity agnostic interface for the Internet.

The scenarios are much more complicated now. The players and stakeholders are numerous with varying geopolitical and economic influence. It is difficult if not impossible to reach a global consensus on even the simplest of issues [1]. How do we accommodate such a world view into a singular networking paradigm? We believe that the solution lies in an architecture which lets these issues play out, i.e provide an approach with modularity and flexibility to accommodate the changing landscape of technology and policies. An architecture providing for the 'tussles'[7] to play out. For instance, the same Service or functionality provided by the same business entity might need different policies at different locations, even if it is implemented on a uniform Service delivery platform such as the Internet (as can be seen in the following articles [188–190]). These requirements make implementation on policy and business rules significant and necessary. Isolating various aspects of the network architecture becomes increasingly obvious with such emerging requirements.

5.1 Benefits of ROSA

Our architecture is conceived with exactly these parameters and requirements in mind. Simplicity is another major consideration along with providing mechanisms to handle different domain knowledge through a uniform interface. For a successful new proposal, the requirements are not restricted to the technological merits alone. The decisions for widespread acceptance of a certain technology is influenced by policy makers, developers and suppliers of related infrastructure, content (and solution) providers and even communities that involve in practice of the related technology [187]. We can discuss the merits of our approach for these groups to identify the acceptability of our architecture.

Basic principles we consider to propose the new architecture include:

- A layer-less architecture composed of Services.

[1]See articles on the great firewall of China, Extra judicial Wiretapping, Blackberry messaging Service controversy in India etc. for a range of issues

- Communication Services, Enabling Services, Application Services etc are all handled equally.
- Domains containing a number of entities administered by a coordinating entity.
- Providing mechanisms to handle/interface different standards used by different domains.

Benefits for stakeholders and players

For policymakers, the uniform SOA based approach provides a coherent vision on how to integrate systems to support organizational and cross-organizational processes to enable effective policy implementations on required functionality. A domain centric view (as followed in our architecture) facilitates communication of technical policies via a single standardized manner, provide an efficient centralized policy repository, eases its enforcement and provides monitoring facilities with quick implementation of policy decisions. The Service oriented approach for functionality also supports planning for technical and interoperability specifications and standards development. The framework we propose can be used to document the specifications and standards required ti support the network architecture of practice. A Service oriented view of the domain also helps to identify activities, business models and gaps, providing focus for decisions about priorities and resource allocation.

Our architectural framework also provides advantages to businesses and organizations that deliver and manage network services, infrastructure and content. The Service Oriented approach is in line with how most of the enterprises implement their business processes, which in turn is derived from their business models. This is true for Services providers and traditional infrastructure providers[1]. The framework of Services we provided are an indicative set of Services (Candidates) and the framework works similarly for Services determined by other domain business processes and implemented.

[1]The traditional infrastructure providers are within the scope of Service providers with approaches like Service Oriented Computing (SoC) and Computational Grid (CG).

These Services can be reconfigured to meet changing operational requirements to adapt to organizational changes without a fundamental reworking. This is a major advantage when it comes to tailoring Services provided by a domain to various aspects of a peering agreement with another domain. For Service providers, our approach provides a modular technological base. The specific advantage of this framework is to enable the development of composable modular and flexible systems where the individual components can be added or replaced more easily than traditional models and where an entire new system or sub-system can be composed from the collection of available Services.

For domain administrators (and policy makers), the collaboration among domains is made easier through defining the behaviors and processes which are needed to share information between domains. Our architecture also makes sharing of applications or functionality easier, as it will be simpler to define Services which can be deployed to meet the common needs of each domain. These in turn provide better returns on technology investment. Besides, Services can be developed or acquired as needed, which means that only those parts of the systems that really need to be changed are replaced, retaining the rest of the systems, thereby reducing both purchasing and implementation costs. In our architecture, leveraging standardised interfaces and component behaviors, domains can integrate with each other regardless of source (location) thus allowing organizations to choose the most suitable application for their purposes.

Our architecture also enables the faster deployment of technology as Services are independent entities. It will often be easier to deploy new Services as long as the needs of the new Services are compatible with the existing interfaces. Even where this is not the case it may still be simpler to alter or replace other components to supply the requirements of new systems. Well defined behaviors and interfaces allow software components to be developed independently, and ideally these components are reusable granular building blocks. Services are defined via their behaviors and interfaces and thus can be used without knowledge of the internal workings of the component providing the service, allowing components to be replaced without causing widespread disruption. Since Services have agreed interfaces, applications can be more easily implemented at another organization or utilized by other applications in the domain, supporting transportability. The proposed framework can be used to provide a Road Map that allows organizations to implement their solution in stages. Organizations do not need to implement infrastructure all at once if modeled using a our Service Oriented framework. They can choose to implement within a heterogeneous environment the standards based interactions that are appropriate for their infrastructure. Legacy systems can be integrated into a Service Oriented infrastructure by providing a Services layer on top of the existing applications, protecting legacy investments.

Migration towards new capabilities and Services

With standards based interfaces, the capabilities of Services can be extended by adding newer capability descriptions to the interface description. Completely new Services can be added or even replace existing ones by publishing them in the registry or making them available within the domain (if the capability for auto-discovery exists). In real-life situations, the standards used will have certain amount of disparity associated with them. A mechanism is needed to address this issue. Besides, in the spirit of bridging across domains, even domain with proprietary interfaces might need to be incorporated, though actively discouraged. one option is to provide for a schema/policy database and translator Service to convert the proprietary formats into standard ones. This Service could be inside the domain,a third party Service or a Service provided by the domain implementing the proprietary standards and data formats.

Versioning is still an open issue, since there are not widely adopted standards to accommodate this. A destructive change (like a domain reconfiguring its business model and Service portfolio) can still be accommodated as the point of first contact remains similar, but the capabilities advertised will now be different. Existing users might need to be updated via Service or domain status updates where appropriate. Older functionality can be made available in this architecture by encapsulating it using standard Web-Service interfaces and describing its data formats in using standard specifications. This might need a Translator/Broker pattern to be used to convert Web-Service requests into proprietary ones. This is an inefficient and expensive proposition in the long term and hence will encourage the migration towards standards based Services in the subsequent iterations.

Any proposed solution (as ours) must allow for transition from past to present and present to future architectures. From the implementation of Web-Services in Enterprise application integration (EAI) , their success and usability of the associated standards, we can infer a pattern to accommodate past architectures with the new approach and upcoming architectures to relate to the our proposal. In our proposal, the focus is on domains and Services, a concept basically present in all communications transactions. Our architecture merely mandates the use of the standardizing of these concepts and tie them together using relationships. It is possible to fit any architectural paradigm within this approach, either natively supporting the standards or via mediators. The requirement is a transport protocol to reach entities and this forms the homogenizing factor for our architecture. The transport protocol would be RESTful [191] (like HTTP) or XML based (like SOAP). How the nodes are identified and routed to by the underlying infrastructure is beyond the influence of our proposal, but we allow tussle to form a consensus to provide an efficient system.

Virtualization of Services

The principle of decoupling of Service interface and capability from Implementation and location extends itself to accommodate other requirement like virtualization. A domain can present a virtual service among its capability, when the Service itself is not Physically implemented within its administrative control, but is still capable of invoking it. Such capability facilitate the provision and transparent usage of remote Services within a domain, with minimum resource provisioning. This approach will however require policy enforcements and long lived SLAs.

Benefits to Communities of Practice

Our architecture supports network functionality diversity. It becomes possible to support a diverse set of Services and business models through providing the capacity to configure the low level elements of the network architecture. The Service Oriented framework enables domain driven implementations by exposing modular processes as separate Services, which can be configured in multiple ways. The construction of technology solutions can be driven by network functionality and policy imperatives implying faster response to community needs. New applications can be created by combining existing Services in new ways in response to community needs allowing for shorter times between identification of a requirement and implementation.

The Domains and Services based approach also allow configuring and re-configuring network architecture based on current requirements and provide agility to functionality. Domain and Service security policies and algorithms can be updated quickly in response to a spreading threat or a newly identified flaw in the existing implementation. For example, a DDoS attack on a particular domain can be mitigated by spreading the information on the incident among collaborating domains, delegating part of the responsibility (filtering, verifying client identities, load balancing etc) to its peers. On demand virtualization of Services can also be a Service that can be provided, which enables other domains to meet SLA criteria in case of unexpected of unprecedented demands.

Improving Security

Security is a holistic approach and not a single implementation which can be accommodated in a particular layer or Service. It requires the coordinated functioning of various factors. A Service oriented approach to security helps to add layers to buffer threats as needed. Apart from a single domain which can implement security policies widely, each Service by itself can mandate specify and implement its own security protocols and requirement thereby owning part of the security responsibility.

A domain security function or Service can be implemented to coordinate and enforce domain wide security policies and requirements. Other individual Services within that domain to wanting to use Services pertaining to the domain can be mandated to refer to this specific Service for security functionality. for example, this Service can enforce data encryption on all outgoing content or mandate DRM solutions to protect sensitive data. For mission critical and highly sensitive domains, a sandbox Service can verify the correctness and validity of external Service requests before processing by actual domain Services. By providing the ability to delegate functionality like security across Services and even domains, a much more mature and flexible solution can be adopted.

Apart from security, the issue of trust among domains is another parameter that affects offer and demand in an inter-domain scenario. Subjective trust and *trust in others*[1] could be criteria for assigning trust to other accessible Services [192]. But, unlike security trust provisioning can be implemented as a Service within a domain, although with the current state of the art eventual bias for positive ratings, unfair ratings, and the variations of quality between ratings can occur [193].

Accommodating Mobility and QoS

Mobility, similar to security, is not a single layer or Service problem. This is one of the main reasons why most of the proposed engineering solutions fail to catch on in layered architectures. Service oriented approach and domains services can provide a modular framework to delegate the various requirements to accomplish successful mobility. Under our architecture, various types of traditional mobility scenarios like node, personal, network, session, etc can be reduced to permutations of *Domain Mobility*[2] and *Service Mobility*. The solutions to these kinds of mobility and their associated requirements need

[1]If company X knows that a service is being used or was positively rated for company Y, whom X trusts, the reputation of that service would increase from the point of view of X.

[2]Domain Mobility occurs when a domain has to sever its relationships with existing peers due to a change in context such as network connectivity domain-wide policy enforcements

to be explored further, but appears to be a much more simpler problem to work with at domain/Service abstraction rather than link/network/transport layer abstractions.

As mobility compared to IP networks can be improved by adopting HIP by tweaking the architecture, our approach provides much more flexibility to implement mobility solutions. We do not restrict the solution into a single solution, but take away the restriction of layers within the architecture this enabling competition among mobility solution proposals. It is straight forward to accommodate proposed mobility solutions for IP based networks within our architecture, but we suggest a revamped look since our architecture offers much more flexibility.

Scalability

The architecture by itself is scalable, since functionality can be aggregated at Service, domain and inter-domain level. Other design patterns like gateways and brokers can be used to improve scalability and abstraction. The flexibility in selecting a subset of the Service Inventory (or Service Candidates) to implement domain functionalities lead to very flexible architectures. For instance, limited capability devices need only implement a limited set of additional Services apart from mandatory Services, as is the case with special function domains (routers, sensors etc). The available number of Services will scale dramatically based on open standards and competitive boundaries. Scalability becomes an significant issue when it comes to composition and management. Solutions have been proposed to address global Web-Service management [194] based on models and industry standards.

A clear study of the drawbacks of our proposal needs to be carried out, to propose improvements to the architecture at initial stages. Coupling of numerous independent services which communicate among themselves locally or over a connectivity link will generate a large quantity of messages. The quantity of messages increases significantly with the number of services utilized. This imposes a limit on the modularity of services, as smaller services implies greater flexibility but with much higher overhead traffic. But, this overhead can be mitigated using aggregation and broker patterns within SOA. For instance, in our architecture we position RM as a central coordinating Service and thus an aggregation Service for the domain. In the broker mode (RM-B), RM becomes the only service that each of the other services interact with, thus reducing inter-service messaging. The downside, however, is a significant complexity in the RM to correctly fit the numerous services into a logical solution for different contexts. Besides, it is possible to apply the SOA broker pattern independent of the RM, where a domain Service can act as a broker Services for a number of atomic Services and provide a higher abstraction of their functionality to applications and other Service users. Thus the amount of messaging for users is significantly reduced.

Efficiency for small set of services compared to legacy architectures will be less, using our approach. However, the advantages of SOA will be inherited into the proposed architecture. The integration of components within heterogenous environments or dynamically changing component configurations is best addressed using our architecture. SOA and Web-Services offer potentially significant benefits to large service sets that undergo frequent change and facilitate reusability. Currently, the XML footprint and parsing cost at both ends of a message exchange does take up time and resources. With high performance as the criterion, Web-Services might not be as efficient as current architectures. Binary XML for interchange can improve the performance, but it is yet to be standardized.

5.2 Open Issues

Our proposal is divided into two different architectures as mentioned in the previous chapter: A simpler first generation ROSA network architecture and a next generation ROSA. We indicate this division because the technologies and standards that are required to implement a fully functional ROSA framework is not yet available or mature enough. But, the fundamental approach for both these proposals remain the same. Thus, the first generation ROSA can be migrated to a next generation ROSA by adding the additional control Services required and reconfiguring the domain to make use of them. We discuss some of the open issues existing in the state of the art today which poses challenges to the implementation of a next generation ROSA architecture. Some of them are relevant to the first generation ROSA as well like the standardization, Service evolution management and domain ownerships.

Evolution of Service specification standards

Many different standards have been proposed and various approaches have been taken to create a widely accepted and usable service composition platform for developing composite Web-Services. Web-Services composition seems to have higher chances of success compared to traditional composition middleware, due to the standardization efforts that have taken place already and widely being used in enterprises and business scenarios. With WSDL, Web-Services may be described in a consistent way according to their functionalities. The standards and specifications we propose for Service and interface descriptions are Web Service (WS-*) standards. These standards have been widely used in enterprises and proven. However, as a requirement for new parameters (like support for semantics and better support for RESTful WebServices) emerge, the standards evolve and are updated. For example, the WSDL standards is in its third iteration, from WSDL 1.0, through WSDL 1.1 to the current version of WSDL 1.2 (renamed as WSDL 2.0). Although some of the changes might be minor (such as renaming PortTypes and Ports in WSDL1.1 to interfaces and endpoints in WSDL 2.0), some changes to standards are major improvements or fixes to existing problems. This creates inconsistencies in the specification on services. However, it is more easier to deal with this challenge in our architecture due to the interface and implementation separation inherent in the SOA approach. But, additional steps like versioning and change management might be necessary when existing specifications standards are updated and used.

Maintaining a current Service inventory with updated specifications, interactions, non-functional attributes, and internal changes invisible to the outside world is a necessary and nontrivial extension to this challenge. When services are updated or changed, the utility and perspective of that Service for its users change. Information like who is manages the changes, who propagates them and the new Service characteristics must be provided for Service Users for them to related to the new Service. All these considerations affect issues such as compatibility and versioning of Services which are part of the ongoing research work in service composition and evolution

A discussion on Domains and ownerships

Since our *Concept of Domains* is a novel approach to composing networks, the definition of domains need to be further exercised against the current scenarios which impose overlapping of administrative functionality. While it is a straightforward exercise to model the meeting of domains for clearly demarcated domains, this approach becomes complex when considering domains which has overlapping, multiple of shared domain boundaries. For instance, a mobile device with the capability to use a GSM service provider to make calls will need to have the network provider mandated SIM card to be used in the device. This SIM card and the Services provided by it is within the domain of the network operator, which it is physically within the domain on the device owner (and hence have some administrative control over it). This scenario projects a shared domain model, where a Service within a domain delegates part of its control to an external domain, i.e, each domain can include the Service within its Service repository but with limited administrative rights and strict policies. It is possible to define a new class of Services to indicate this shared nature, but we believe it is easier to extend the current model of domain Services, but with more constrains for domains on Service ownerships.

Semantics and Ontologies

As automation becomes necessary in stringing together Services to extract more useful functionality, the methods used to describe Services becomes critical. Besides the Service capabilities and Interfaces expressed in standard formats, it is also necessary to specify the effects and characteristics of the Service to models its functionality. This requires accurate semantic descriptions of the Service to be available. However, the field of semantic languages and domain ontologies are still an ongoing research area. Standardization is slow and widespread use of these technologies outside of academia and research projects are few. Standardization of domain ontologies in necessary to enable automatic peering between domains, which in one of the fundamental design tenets of out architecture. This would enable domains to agree on and understand what various terms mean in a global scale. Including a rich sematic language based description to Services also facilitate more accurate search and matchmaking capabilities for ROSA, together with increased capability to inspect and process user requests. It is also possible to classify and compartmentalize similar Services within the inventory based on their inputs, outputs, behavior (semantics) and other parameters which are not possible to be expressed semantically. More accurate provision of QoS is also a derivative of sematic models of a Service. The sematic models are also quite useful when Services need to be dynamically replaced at runtime with minimal effect of the remaining processes[1].

Automatic Dynamic Service Composition

Our future architectural proposal functions with an automatic dynamic composition of Services as its fundamental requirement. We assume that Services are fully described with semantics covering all aspects of its behavior. We assume that such technologies will be available to combine selected Services based on domain policies and contexts by

[1]A replacement Service with similar input/output/parameters with a similar semantic behavior is much more suitable for replacement that just syntax match

a Composition generator to produce a composite Service which can provide a meaningful functionality to satisfy a query by a Service user. Both automatic Service composition (composition without human involvement at any stages) and dynamic composition (where Services are orchestrated on demand, based on the incoming Service request) are both fields of ongoing research, with focus changing towards semantic representation, composition description and composition generation. Until a reliable and proven solution emerges to this end, the capability and scope of the proposed future ROSA will be limited. But, the fundamental approach remains the same with more capable composition related control Services replacing the currently existing limited ones.

This network architecture based on a dynamic runtime service composition requires much more improvement in dynamic composition methods and related verifications. The idea of automatic composition of component services to form a composite service to solve a particular problem is not a new approach. These methods are widely used in enterprises and businesses in Web-Services. Most of the composition are done manually and the correctness of the functionality of the composite service is done during design time and manually as well. But, when we look at our architecture, and the idea behind compositions (relationships), there could be potentially thousands of instances of services that could be utilized. To retain flexibility of the architecture, a capability must exist for these services to be chosen and composed during runtime based on constraints and relationship metrics. Under such a scenario, dynamic and automatic composition of services are essential.

We introduce the need for verification of the correctness of these compositions in real-time as well. The composition of a composite service can be based on a process description, a model or a requirement. The end result of the composition is a solution logic and a list of services to be used. But, this solution depends on the intelligence of the RM and the composition logic used. It is imperative that there exists a mechanism to verify the correctness of the composition (at runtime) so that any potential errors or exceptions can be inspected and mitigated. One proposal is to use formal methods to model and simulate the composition, and verify the correctness. This can be done using a dedicated service within the domain or a third-party service. While most of the common tasks will already have verified composition solutions available, the capability discussed here will add more reliability and confidence in the obtained service compositions within this architecture.

An extension to this challenge is the verification of the generated Service compositions. In mission critical and production environments, it is important to verify the correctness and validity of automatically generated compositions before they are invoked to solve a specific problem. This is also an ongoing research topic, with many approaches being explored. We bound this challenge within the Composition Verification Service, and leave it to the tussle for an efficient and accepted implementation.

A security framework

With Service interfaces as abstraction, we have argued that security can be accommodated as necessary within the individual Services, and the cumulative policies and constraints contribute to the overall Security of the architecture. This approach does not go far enough to provide a uniform security solution for our proposal. A security architecture for ROSA is needed to enforce security parameters at Service and Domain level to coordinate, enforce, monitor and maintain the security component contribution by individual Services within a domain. A single *Security Service* is not the ideal solution here, as security is not a single Service characteristic.

5.3 Comparison with other proposed architectures

Many alternate architectures have been proposed along similar veins as our proposal, out of the identified need for a improvements in the current approach. We have covered some of the proposals in chapter 2. Two of the notable proposal include Ambient

Networks (AN) and TurfNet[168]. These proposals provide a much more comprehensive approach for a new network architecture than other projects that address a subset of concerns such as HIP, DONA, TRIAD etc. Many of the new proposals, in an effort to bridge heterogenous networks, propose varying approaches, some as far as mandating no common standards or even interfaces. TurfNet [168], for example, assigns complete freedom to domains to act as they wish within their administrative boundaries also conversions among TurfNets. This solution leads to immense overhead and over reliance on gateways to translate or act as proxies between communicating entities. This is not a suitable approach with regards to scalability or interoperability.

The TurfNet proposal revolves around the meta-architectural principle that different regions of the network should be allowed to differ from each other:"minimize the degree of required global architectural consistency"[195]. Our contention is that too much diversity leads to fragmentation, incompatibility and lack of interoperability. Our proposal strives to accommodate diversity but is careful enough with standards not to promote it with incentives. In terms of scalability, TurfNet's bottleneck lies with the explicitly defined gateway nodes that relay inter-TurfNet traffic. The relay method is compounded by state explosion in gateway nodes[1]. Further solutions are needed to address performance or load problems associated with address translation of all inter-Turf communication.

In AN, the framework is divided into a modular control space (called Ambient Control Space or ACS) and a configurable user plane. The ACS is proposed to allow for the plug-and-play rearrangement of management and control functions, while the user plane provides enhanced data transport over a wide variety of connecting infrastructures. Our proposal agrees with the motivation behind AN [35] and the aim to build a "*middleware that hides most of the network complexity, as well as the aspects of dynamicity, to maintain and sustain performance and usability objectives as the overall system develops*"[196].

[1]These can be mitigated to a certain extend using state aggregation mechanisms.

The AN implements its control functionality (ACS) using a collection of Functional Entities (FE), and differentiates various interfaces that it offers. The Ambient Service Interface (ASI) towards applications and Service, the Ambient Resource Interface (ARI) towards basic connectivity and communication between control space and Ambient Network Interface (ANI) towards other dynamically connecting networks [196]. In contrast, our proposal provides a simpler and more uniform approach for interfaces. The control functions in our architecture are implemented in the same fashion as user functions and invoking either of them also follows the same pattern. We do not differentiate among Services in descriptions or invocation methodology. The differentiation presents itself with the functionality of the Services. The control space (to borrow the term from AN) in ROSA consists of a set of mandatory Services (Domain Control Services) that provide domain specific control functionality, centered around the RM while additional functionality is provided by other Services.

Similar to AN ACS, the ROSA control framework (RMs + mandatory Domain Control Services) are modular and support 'plug-and-play' where additional additional control functions can be added and existing ones modified on the fly. This makes ROSA control framework distributed and modular. Going through the list of AN control functions and managers, we observe a similarity in the functions proposed, but we follow a much more uniform approach. Similar to AN, policies play a critical role in our architecture too, within and among domain interactions. AN focusses on a network of networks, where a device form part of a network, which can further merge to form larger networks. In our architecture, the building blocks are Services and domains. we expand and contract the scope of these definitions to fit anything from small sensors (or a single function device) to enterprises (with thousands of Services) under the same paradigm.

5.4 Drawbacks

Although the ROSA architecture can accommodate legacy network architectures through patterns like brokers and wrapping, it functions efficiently and naturally with a transport level delivery mechanism that need not be further broken down for processing for other restricted layered architectures. For instance, the domains and Services fits into a hierarchical addressing scheme and can be identified by directly using naming conventions like URI. Inter-domain routing is the most efficient when domains can interpret and route these instead of breaking it down further to IP-addresses. For our architecture to achieve full potential, new routers (routing domains) and schemes are needed that can route flat-label and URI-based communications with high efficiency and performance.

Coupling of numerous independent services which communicate among themselves locally or over a network will generate a large quantity of messages. The quantity of messages increases significantly with the number of services utilized. This imposes a limit on the modularity of services, as smaller services implies greater flexibility but with much higher overhead traffic. ROSA is a message based architecture. When such an SOA based architecture is composed, signalling and traffic load will be higher than legacy network architectures. Besides, the main phase of composition is the establishment of a relationship between domains, where negotiation can produce substantial signalling overhead.

In ROSA, communication and application Services are now a part of a layer-less architecture leading to new types or security challenges and potentially higher vulnerability. The Web-Services security proposals can mitigate these challenges to a certain extend and, theoretically at least, the argument that Service level security can vastly improve the overall security. A ROSA security framework still needs to be proposed to outline and enforce a fundamental level of protection to the architecture compositions. A set of security protocols and mechanisms are to be identified and specified as a part of this framework.

The idea of stateless network as envisioned by the inventors of the original Internet is blurred in our architecture. The ROSA network is not stateless anymore as it relies on 'relations' that are based on computing a wide range of past and present parameters such as history and trust.

CHAPTER 6

CONCLUSION

It is hard to predict the future of the Internet architecture, as it was hard to foresee the success of the IP based Internet. The design has its roots in an early academic project. The dynamic evolution of network technologies implies one constant factor that could be safely predicted - and that is change itself.

With advances in technology came new application scenarios which further fuels the demand for technology innovation. In a connected world, the network architecture has to keep up with the evolution of applications and new technologies. The current Internet architecture has well-known shortcomings in flexible naming & addressing, security, mobility, QoS, real-time.

It is safe to assume that the network architecture of the future must be flexible, scalable, evolvable, configurable and dynamic to facilitate new applications and new infrastructure scenarios. Our goal with this work was to contribute to the research for a new Internet architecture and lay down a set of grounding principles to overcome current Internet shortcomings.

Past attempts to find engineering solutions to the above challenges by blurring the layer boundaries (cross-layering) and adding on functionality (like security solutions) have resulted in a complex, sub-optimal, fragmented and inefficient outcome. Extreme cross-layering without a uniform abstraction leads to complex and ultimately inefficient designs.

We propose a layer-less, 'Service Oriented (SO)' architecture approach as an alternative to the existing layered paradigm. In our proposal, called Relationship Oriented Service Architecture (ROSA), we identify services as the basic building blocks of the architecture. We abstract communication and network resources in the same manner as any other resources to provide a high level interface to applications and Services.

In ROSA, we define '*domains*' as units of administrative control and logical space for Services to be hosted, discovered and be used. In order for domains and their services to cooperate, we established a 'Relationship Manager' (RM) as a well-known standard contact point present in all domains. The RM orchestrates domain Services and is responsible for initiating, controlling and managing domain and inter-domains communications. A RM is not a stateless entity as it computes various parameters such as trust, cost, QoS, reliability, security, policies and others before activating a relationship between services.

We outlined an initial set of Domain Control Services to accomplish overall control and management functions. The RM manages associations among Services within a domain, across domains and provides a high level abstraction to applications. In chapter 4, we outlined the mechanisms, standards and communication models necessary between domains to interact with each other.

The need for a new architecture is also recognized by other research groups, who have proposed modifications and '*clean-slate*' approaches for a new Internet architecture. Notable ones include Ambient Networks (AN), TurfNet, Haggle, DONA, HIP etc. DONA proposes a disruptive redesign in Internet naming, with the fundamental proposal suggesting a move away from the host-to-host paradigm to a content (data) centric approach.

However, DONA fails to address other architecture related shortcoming in the current Internet like security, QoS etc.

Haggle adopts opportunistic connectivity, data/meta-data management and sharing for applications. Although Haggle follows a layer-less architecture, there is no standardized abstraction proposed. This leads to increasing complexity as utilization grows. The initial scope of 'seamless connectivity for mobile devices' is not wide enough to accommodate all emerging requirements.

While the separation of names and addresses in the Internet architecture is long overdue and could resolve some of the challenges, it still cannot address other architectural limitations imposed by a layered abstraction. These limitations are simply ignored or otherwise magnified. 'Ambient Networks' approaches these challenges with a comprehensive redesign of the architecture. The main focus is on seamless interoperability between heterogeneous network domains. The original proposal to ignore IP-addresses as the network layer technology (due to its shortcomings) was later changed with the acceptance that IP might prevail after all[1].

Ambient Network proposes three kinds of interfaces - Ambient Service Interface for applications, Ambient Resource Interface to abstract connectivity layer and Ambient Network Interface towards other networks to communicate and compose. All these interfaces are controlled and managed by the Ambient Control Space (ACS). In our architecture, we do not make this distinction. We propose a simpler and uniform approach to treat all resources (connectivity, application and control) in a similar manner, utilizing accepted and proven standards. It is interesting to note that the prototype implementation of the AN used for validation [197] used Web-Services as the front-end for FEs in the ACS, substantiating our argument that SOA with WS as interface standards are a viable option for accommodating any architectural implementation.

TurfNet is another notable proposal built on the idea of network composition. However, we do not find it generic enough and TurfNet lacks the clear foundational guidelines

[1]See papers [35]and [183]

as SOA or any others. An ultimate aim of all these approaches is to raise the level of abstraction to shield the applications from low level communication details.

The Service Oriented based approach maintains a separation between interface and functionality to fully exploit the advantages of Service Oriented Architecture. The ability for Services to be location independent, modified, exchanged or replaced (even during runtime) adds flexibility to our proposal. A further advantage is that the architecture will be able to integrate different QoS and mobility methods into one solution.

A major concern for any clean slate approach is the ability to accommodate or migrate from legacy architectures. This is a major challenge to all new proposals. A new architecture must offer migration strategies to increase acceptance, and to reduce technology and business risks. We recognize the fact that IPv4 (and IPv6) will be around for a long foreseeable future, but also adopt the view that other approaches besides IP-based naming/addressing/routing schemes will emerge. There are already indications of such evolution in HIP and other flat naming proposals. The architecture we proposed can accommodate legacy architectures via various patterns like wrapping, gateways or encapsulation, analogous to how the migration is currently occurring from IPv4 to IPv6 in the networking world.

We propose two architecture instances offering milestones along a path to replace today's legacy architectures, and to bridge gaps until certain new technologies like dynamic service composition are available. The First Generation ROSA is a version that is implementable now. This version relies more on pre-configured services and configurations. The Future Generation ROSA is an automated, dynamic version of our architecture based on much intelligent Services and emerging technologies.

Many basic mechanisms still need to be studied to arrive at the full scalable and dynamic future ROSA. Examples include representation of domain knowledge in machine understandable manner, understanding relationships in a semantic web, automatic and dynamic Service compositions, validating and verifying the correctness of potential composition candidates, control signalling and peering in a distributed multi-agent system etc.

Further research is needed before standardization bodies such as '*OASIS Web Services specifications*' will take them into account. Nevertheless we hope to have shown how important these technologies will be for the evolvement towards a future Internet architecture, and that network and service architectures should be integrated into one common, overall solution.

NOMENCLATURE

CIDR	Classless Inter-Domain Routing
DDoS	Distributed Denial of Service
DPI	Deep Packet Inspection
EAI	Enterprise application integration
IRTF	Internet Research Task Force
MANET	Mobile Ad-Hoc Network
NAT	Network Address Translation
PDDL	Planning Domain Definition Language
PPM	Polymorphic Process Models
QoS	Quality of Service
RDF	Resource Description Format
RSVP	Resource reservation protocol

SCTP	Stream Control Transmission Protocol
SEA	Service Execution Agent
SIP	Session Initiation Protocol
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
WSCI	Web Services Choreography Interface

APPENDIX A

5^{th} GENERATION NETWORKING PRINCIPLES FOR A SERVICE DRIVEN FUTURE INTERNET ARCHITECTURE

The following article by the thesis author appeared in *Springer Journal for Wireless Personal Communications* dated *September 2010* [174].

A.1 Abstract

The vision of all-IP networks where IP forms the simple common layer understandable across the whole network has undeniable advantages. However, such simplicity comes as a major hurdle to flexibility and functionality to the architecture. This is evident from the increasingly numerous and complex engineering solutions and optimizations

required to accommodate essential qualities like mobility, security, realtime communication support etc or to mitigate the shortcomings inherent in the 'traditional Internet' architecture. While a clean slate approach to address these shortcomings is not an option in a realistic scenario, it is important to examine the architecture as a whole to address emerging network requirements and overcome existing shortcomings at the architecture level rather than engineering solutions to an existing inefficient one. This architectural re-examination should also facilitate discussion into what design principles for future generations of Network Architectures which will eventually replace the design tenets for the current Internet. While 3G and 4G systems were more focussed on convergence towards an All-IP network and some improvements in the core network, the architectural design remains stagnant with layered paradigms and inherent inefficiencies. A departure from this shackled approach could be the distinguishing feature of 5G systems and beyond.

We claim that there is a pressing need to move towards a Next Generation Network (NGN) architecture built to natively support requirements such as network resource abstraction, mobility, security, enhanced routing, privacy, context communications, QoS, parallel processing, heterogeneous networking etc. Instead of treating the network as just providing connectivity specified by endpoints, it is of great advantage to applications to recognize it as a service *characterized by attributes, abstracted to a higher level to represent a collection of capabilities that the network offers. This uniform high level abstraction can effectively mask the heterogeneity and implementation discrepancies in the underlying infrastructure. Besides, in a network environment where an connectivity instance might transverse diverse business/ownership/capability domains, the approach proposed in this article can provide a transparent abstraction for resource negotiations across the domain to be available for end-to-end setup. This architectural change should also be manifested according to the principles of SOA to ensure interoperability, backwards compatibility and migration. In this article, we introduce a Service Oriented framework and network architecture aimed at tackling the heterogeneity of emerging requirements and proposed solutions into a coherent interoperable architecture using*

Web Services specifications as the basic standards. We propose propose t o model the new architecture on relationships between entities and discuss the motivation this new architecture in the form of a new framework called ROSA.

A.2 Introduction

The Internet, as a network of connected computers, came into existence in the 1970s [198]. The early Internet interconnected a stationary set of nodes. The architecture and the protocols used were designed to accommodate and exploit this simple stationary nature. The designers wanted to build a network infrastructure to interconnect all computers in the world together and provide a framework for yet unknown applications to be invented and run [43]. The nodes were well described by IP addresses that identified the nodes directly on the network. Routing protocols then took advantage of the static nature of the Internet.

Although the usage and possibilities of the Internet have expanded beyond its initial scope, the design principles and architecture of the original Internet are still followed today. The success of the Internet has by itself shackled the possibility of any dramatic change or a completely new architecture from being implemented to accommodate the requirements that were not envisioned in initial design stage. The massive installed base of routers, clients and other network equipment supporting today's network infrastructure makes sure that any significant changes to the architecture of the Internet protocol (IP) based network will be overlooked, if not ignored. The financial aspects of migration will play an important part in migrating to any other replacement proposed from today's architecture.

Due to the explosive evolution of the Internet into what it is today, it is difficult to clearly define 'the scope of the Internet' or specify 'edges of the Internet'. The boundaries of the legacy internet (if we can call the earlier iterations of the Internet that) is continuously being blurred by the introduction of new devices, violation of the original design tenets and the incorporation of new paradigms. Currently, a typical view of the Internet implies *an electronic communications network that connects computer networks and organizational computer facilities around the world.* This is a very generic definition which makes the term 'Future Internet' a very wide area for research and a 'Future Internet Architecture' a generic network architecture that could address all networks.

A.3 The need for a New Architecture

With time and technological advances, the networking solutions have been steadily increasing in complexity. To accommodate new requirements, the Internet has been engineered with more powerful routers, faster backbones, faster processing at end points and traffic shaping to accommodate new models and better performance. These often incompatible engineering solutions (or 'hacks to the original stack') provided temporary fixes to new problems but introduced unnecessary complexity, a few of which will be singled out in the later sections. There exist fundamental inefficiencies in the current network architecture which cannot be addressed efficiently. For example, challenges with mobility, end-to-end Quality of Service (QoS), security, trust etc. In implementing the work-around, the Internet today breaks most of the design principles initially conceived for it [9]. Some examples can be perceived from 'end to end' principle violated by middle boxes, NATs etc. [10]; fairness restricted via traffic shaping, packet inspection; best effort breached with overlays; stateless network concept infringed by intelligent middle boxes [11], stateful proxies, label based router etc. The vastness and distributed control nature of Internet today, makes it difficult to implement distributed applications with realtime guarantees necessary for certain types of communications. These changes do not converge towards any new architectures, but are add-ons or overlays piggy-backing on the same old design. Some of the urgent problems like address exhaustion, better compatibility to emerging technologies via header extensions and better security are squeezed into the IPv6; which still addresses only a part of problem, not the network architecture limitations as a whole.

We can find an instance of this complexity in the current Internet security architecture. Security was not as important a concern as openness and fairness during the birth of the ARPAnet. The fact that the network placed no restrictions on connectivity meant that innovative applications could be deployed without obstacles, which essentially lead to the growth of the Internet to the magnitude we witness today. However, the very same design tenet has now made protecting the network from malicious hosts very difficult. For example, while rudimentary security measures solve most of the problems (e.g., security holes in an applications can be patched and end-to-end security protocols can be deployed, or security overlays for specific protocols), the openness has made it difficult to defend against Denial of Service (DoS) attacks. A visualization Fig. A.1, adapted from [3] indicate the complicated and patched security architecture of the current Internet. The lack of a harmonized security strategy and multiple approaches manifest themselves as cross layering and conflicting overlays. The lack of a common trust, privacy

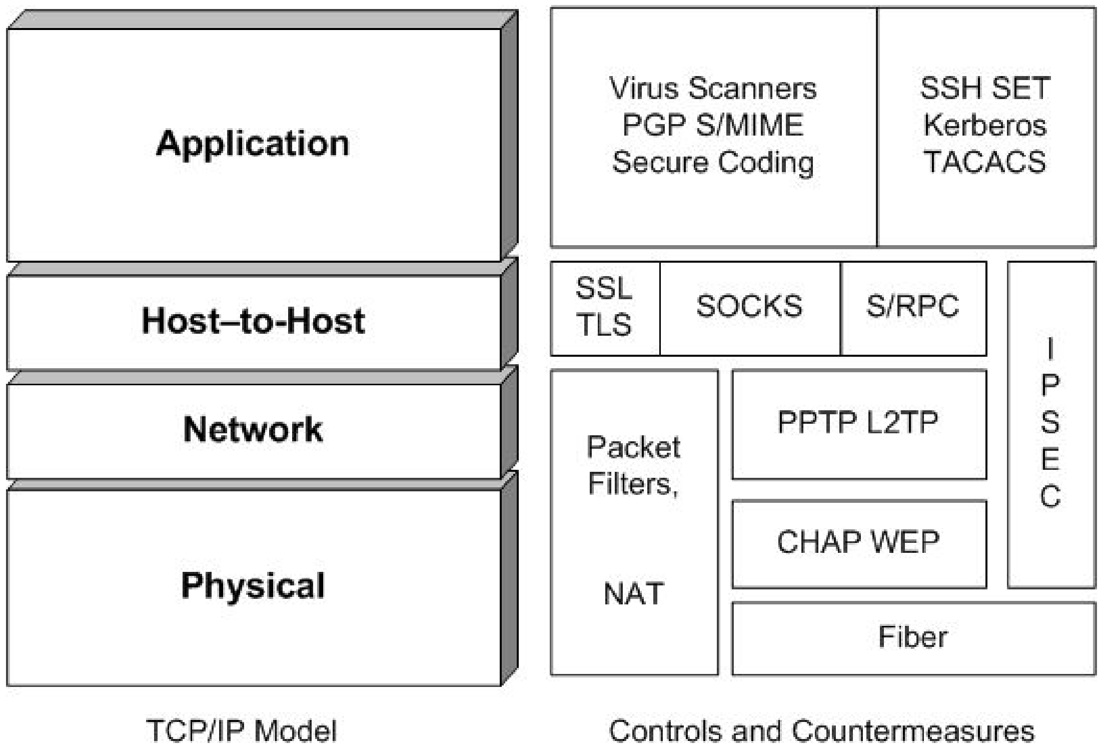


FIGURE A.1: Internet security controls and countermeasure [3]

and security approach is just one of the shortcomings of the current Internet architecture. The Ambient Networks project, a European sixth frame work project identified some of the requirements to be addressed for their next generation communication architecture [35]. Haggle [36] identifies that the root cause for some of the usability deficiencies with regards to mobile devices today arises from the synchronous IP-based APIs presented to applications along with the numeric addresses as end-points. Applications implemented in such models rely on networking infrastructure for end-to-end communication without taking advantage or being aware of local or neighboring network resources. We list a few of the challenges or requirements encountered by applications below:

- Concept of Location/Neighbourhood awareness, proximity etc.
- End to end service oriented communication.
- Separation of identifier and location in naming and addressing.
- Session continuity & management across domains.
- Common trust, anonymity and federated identity management.
- Parameter/Metric based routing (added value based routing).
- Routing facilities based on application layer needs.
- Efficient Mobility, Multihoming, delegation, indirection.
- Capability signalling across devices, domains.
- Real-time and Distributed real-time application requirements like priority, guarantees etc.
- Scalability for trillions of nodes.

As can be observed from the above list, the issues to be addressed are rather basic and spread across the existing 'layers' of TCP/IP model; i.e., it is difficult to solve the above shortcomings at a particular 'layer' of the current Internet architecture. But, these arguments in no way propone demise of the Internet, but suggest the strain impressed on the legacy architecture it still follows. The stress on the current architecture is not limited to the new usages of existing technologies, but also arises from new technologies that are incompatible, but forced into compatibility for legacy interoperability. For instance, ad hoc, vehicular and sensor networks differ dramatically from the relatively static 'client-gateway-server' design of the Internet with the number of nodes stretching into billions and extremely dynamic mobility scenarios.

While the internet has tried to avoid vertical silo (smoke stack/chimney model) effect by abstracting horizontally with layers rather than complete end to end solutions, in reality, the effect has been more hard-coupled. The proliferation of IP as the *de facto* standard for network abstraction has made the generic concept into an IP-Hourglass model [44]. This model has worked remarkably well over the past few decades, but does not perform well when exercised against new paradigms and applications. There is an interesting discussion growing around the 'waistline' of the internet with various opinions and concepts emerging on 'what will or should' be replace or added to expand the waistline of the current network architecture [32, 44]. With the shortcomings of IP (such as the identifier/locator dichotomy), it is only logical to provide an alternate addressing scheme to take advantage of the innovation in the networking and routing domains over the past few years. However, *what is the best alternative?* is still an open question. It is this openness that should be embraced rather than providing a solution which in a few years will find itself inefficient or even unsuitable for use due to newer unforseen requirements. In a best effort packet delivery mechanism like IP, the concept of a separate control channel is diminished (apart from already existing internal and external routing protocols) [44]. Each packet carried enough information for processing at intermediary nodes. Following convergence and adoption of IP as the *de facto* abstraction at the network layer, the concept of the self contained packet became the norm. This simple and common layer was advantageous for interoperability but is manifesting as a major challenge for the flexibility for applications running at higher layers. With the new requirements like QoS and security, extra intelligence needed to be built into routers to examine the contents of each packets to decipher what needed to be done with it for additional functionality (as per end user signalling through RSVP, for example). With the emergence of realtime multimedia as a major part of the traffic on the internet, separate control protocols had to be developed (SIP, RTP) just to facilitate the delivery of multimedia streams/sessions. With QoS, routing is no longer simple forwarding of packets but consists of prioritizing, queuing, dropping, tagging/marking and so on. The routing architecture of the current Internet does not support packet forwarding based on such rich or descriptive parameters.

These trends segment the Internet as a collection of application level networks (torrent, IMS etc), each overlay addressing a specific application requirement or functionality. Different control protocols implemented in a distributed fashion decide the nature of the overlays. As articulated by Aguiar [44], the concept of in-band control signalling through the packets necessitates the 'hard' processing of each packet to provide additional functionality to the flow. This is hardly efficient when the choices become numerous and the packet count follows suit. A separate control plane or architecture might be necessary, independent of data flow to facilitate added functionality to the communication via networks. This will facilitate capability negotiation across different control domains (like businesses, Autonomous Systems etc) for setting up sessions or temporary peering agreements separate from data delivery mechanisms. The proposal to have an additional hourglass model for the control architecture complementing the data hourglass model (with IP at the waist) [44] for networking might be one approach to address this challenge. The idea of separation of concerns (control and data) within the network architecture brings flexibility at the cost of simplicity.

A.3.1 New Paradigms

The Internet evolution has been characterized by ingenuity on the part of software and application designers to circumvent the architectural limitations, and has brought into play varying 'players' into the sphere of influence [7]. The resulting 'tussle' influences not only the direction of the future evolution of the Internet, but also the nature of the next generation architecture like notions on design, space for tussle etc.

Access technologies and applications emerge independent of the Internet, which in due course requires interacting with the network for various reasons. We find powerful mobile communications devices, sensors, medical implants, vehicles and a host of other network capable appliances emerging. Besides, new networking models like ad-hoc networks, vehicular and sensor networks present challenges which are not efficiently handled by the current architecture. The number of connected nodes in the Internet has gone from a few in the early eighties to millions (currently), with a strong possibility to be trillions [28] with the inclusion of ad-hoc nodes, cheap sensors and networked vehicles in the future.

User demands like interactive resources, user generated content, content sharing, local access of distributed content, anywhere/anytime access of own data etc. requires the underlying architecture to support a set of basic capabilities, which are inefficiently implemented into the legacy Internet [29]. The demands can also include demands based on or on behalf of the users by other entities such as governments, corporations and content owners. For example, the open and end-to-end nature of the Internet is broken by middle boxes to accommodate for address exhaustion, security etc [10]. This necessitates architectural changes incorporating such additions.

These are not independent driving forces. These factors tend to influence each other to a stage where the underlying architecture can no longer efficiently support the newly construed paradigms. This will be the case with any new tightly designed architecture. The boundaries of such architectures will be tested. One of the more accommodating architecture would be the one which account for this growth (‘design for tussle’ [7]), or a system modular enough to keep pace with the innovations around it. The idea of service oriented architecture and service composition, manifested in different proposals as network composition is worth considering in this context.

A.4 Challenges to a New Architecture

There are two common methods of approach that can be utilized when thinking forward. One is the *Incremental Approach* aimed at maintaining backwards compatibility while migrating towards a new architecture (E.g. IPv4 to IPv6 migration). The other approach is to have a *Clean Slate Thinking* to fix all the problems that can be identified as being inherent in the current architecture. Architectural changes to the core of the Internet (E.g.: IPv6) and add-on/overlay services (E.g.: MIPv4, MIPv6, IMS etc.) have met varying levels of success. There are a host of new architecturally superior implementations and changes dismissed *a priori* by the marketplace, due to reasons such as ossification of the TCP/IP model and business aspects of bringing about a dramatic change in the currently installed infrastructure base. It is easier for researchers to consider a clean slate approach of a new architecture, protocols and service. However, such an approach is generally unacceptable due the changes required to already existing infrastructure and devices and due to the disruption of existing businesses. This approach, while ideal from a research standpoint is difficult to implement in the current scenario.

The incremental approach to addressing the current limitations is attractive to service providers and network operators in terms of cost and availability. This approach often produces inefficient and often complicated solutions. An ideal solution to this conundrum will be to suggest modular incremental changes to current architecture aimed at addressing immediate problems, which function as milestones or part of the transition towards a completely reconsidered and modular network architecture. The idea of overlay networks (for example, Peer-to-peer overlay networks [33]) is quite relevant in this context, where a new technology can be implemented at ‘present time’ over existing architecture, so as to bring in the new functionality without a radical change to the underlying architecture. This functionality, at a later time can be accommodated as a part of the architecture itself, if designed to relevant open standards.

The above mentioned approach is not only true for functionality overlays like IMS, but completely reworked architectures as well. Consider the migration towards IPv6 from

IPv4 in this context. The newer Internet layer (IPv6) can coexist with the current one (IPv4) via gateways connecting islands of IPv6 routers to the IPv4 world, software encapsulation like the 6to4 transition mechanism [11] or tunneling etc. [34]. In the future, when most of the nodes (routers, specifically in this case) support IPv6 in the future, the IPv6 'islands' automatically become the 'main network', with IPv4 becoming 'legacy islands' interfaced via 'legacy' gateways. This approach however requires a scale, consensus and collaboration from major players in the research community and industry. A study of the current state of the art will reveal that both approaches are being explored by various entities globally, substantiating the necessity and urgency of such a transition [30, 32].

A.5 Existing Approaches

Under the current network architecture, the applications are responsible for setting up all bindings required for communication. This necessitates that the software is written to specific underlying network architecture, without modularity. The close binding also makes it difficult for developers to implement applications and solutions that can adapt to new communication mechanisms. Most of the new 'flexible generic platform' proposed to overcome such limitations concentrate on abstracting the applications from the underlying network architecture, mostly by adding one more abstraction layer over the existing naming system (IP). These approaches, to an extend, mitigate the ill effects from current dual usage of IP addresses as end point identifier (name) and location (address). The reliance of certain applications on the Domain Naming System (email, web addresses etc.) together with the inability of the DNS to adapt to rapid updates make it more difficult in dynamic mobile environments. The migration from IPv4 to IPv6 will provide some temporary relief to pressing issues such as address exhaustion, resource allocation support via header extensions and improved security features. But, IPv6 does not address the architectural limitations of the Internet. There are various proposal and projects at different stages of maturity being considered by the Internet community, industry and academia related to the architectural nature of the Future Internet.

Host Identity Protocol [51] identifies the naming and addressing of entities as the key challenge in today's architecture and proposes as a solution to separate them, decoupling the usage of the address (i.e. the IP address) as the identity of resources or nodes. The separation is achieved by introducing a new layer between the conventional TCP/IP stack between the network layer and the transport layer. HIP uses cryptographic identifiers as the Namespace which helps to integrate baseline end-to-end security into the architecture when used with Diffie-Hellman [52] and appropriate security protocol, such as Encapsulated Security Payload (ESP) [53]. Each node has a private/public key pair and the node's identity is a hash of its public key. Several solutions have been proposed to accommodate mobility and Multi-homing [66, 67] using HIP. However, there are some undesirable consequences where the node loses its identity if the public key is ever changed or compromised. Haggle approaches these challenges using the concept of 'Pocket Switched Networking' [77, 78]) to take advantage of both infrastructure based and Ad Hoc (peer to peer) communications opportunistically. The Haggle network architecture is aimed at providing seamless network connectivity and application functionality in mobile environments by separating application logic from underlying network architecture.

Ambient Networks (AN) [35, 56] introduces the concept of horizontally structured mobile systems that offer common control functions to a wide range of different applications and interface technologies to provide a common networking concept to adapt to varying heterogeneous wireless and service environments. The AN naming architecture adopts a layered naming model, with separation concepts borrowed from layered naming architecture [30] and HIP [51]. Dynamic bindings at different layers enable the basic mobility of nodes, 'bearers' and applications [70].

Data-Oriented Network Architecture (DONA) [79], borrowing heavily from other exercises like TRIAD, SFS and HIP, suggests a clean-slate redesign of Internet naming and name resolution, to address specific features such as persistence, availability, and authentication for service access or data retrieval. DONA justifies this proposal pointing out the existing discordance between historical design (host-oriented) and current usage (data-oriented). However, the clean slate approach is mostly limited to how Internet names are structured and resolved. As with HIP, DONA replaces DNS names with flat, self-certifying names, and replaces DNS name resolution with a name-based anycast primitive that lives above the IP layer. DONA proposes that names handle persistence and authenticity, while name resolution handles availability. There are other attempts to approach the problem from enterprise perspective through Application Oriented Network Architecture (AON) [80], borrowing ideas from NGN architecture [81].

Another approach is to extend today's architecture using middle-boxes (TRIAD) to avoid the migration to IPv6, or base the architecture on a new central parameter or paradigm. DONA and Haggle architectures revolves around 'data' as the center piece. There are advantages as well as pitfalls to all these approaches. For example, when the architecture is designed around data as the most important parameter, then the design must encapsulate and manage all data as haggle proposes to do. This is not an ideal approach as other applications might not want to relegate ownership of their data to middleware. Identifier-Locator Network Protocol (ILNP) [199] proposes an extension of the 8+8 for IPv6 [54] idea and SHIM6 [55]. ILNP specifically addresses mobility and multihoming issues with current implementations, integration of middleboxes like NAT, enhancement of end-to-end security, among other features. Reliance on DNS for location update of mobile nodes apart from ICMP locator updates might be a shortcomings when dealing with highly dynamic and unreliable environments.

There is a significant amount of discussion forming around the EIFFEL project [200] regarding the nature and process of arriving at the Future Internet, albeit from a European perspective. This project, not limited to technological issues but also social issues surrounding a future networked society, is organized around a number of Technical Areas (TAs) focusing on technological, societal, regulatory and policy-related questions. For a project of such magnitude, it will be a while before its future internet architecture proposal emerges, to be studied or compared to existing solutions. The projects mentioned here are not a conclusive list of attempts to address the architectural shortcomings of the Internet. They are too numerous and outside the scope of this article. But, the conclusion derived implies that the research community and industry recognizes the shortcomings and are exploring various approaches to address them.

A.6 A different Approach to Networking

We develop our vision from a top down approach, from the point of view of the application developers. From such a perspective, networking is not just connectivity specified by an end point tuple (as in Berkeley APIs). A network is a collection of distributed *services* that are *available* to the applications. The application developer should not worry about the state of the network at application runtime during the application design. This decoupling of addressing network end points directly in applications can be achieved through a uniform approach to exploit the underlying network infrastructure via a generic, rich and standardized interface. Such an interface provides abstraction of network capabilities to applications and decouples the heterogeneity arising from the below mentioned factors. Adapting to heterogeneity forms one of the basic characteristics of our approach. Heterogeneity in various layers of the current architecture arise from various sources including:

- Network technologies, devices and Operating Systems
- Middleware solutions and communication paradigms
- Programming models and languages
- Services and interface technologies
- Domains and architectures

- Data and document formats
- Nonfunctional aspects such as information models, security, availability, transactions etc
- Business borders
- Communication procedures and security policies

A.6.1 Network as a Service

We claim that the layered paradigm in networking architecture is not the ideal approach for abstraction. As with Haggle[36], we propose a layer-less architecture which abstracts the underlying connectivity and network computational resources to applications via a high level API. Applications should not be burdened with attaching themselves directly with end points or the connectivity status of various available interfaces; it should automatically be taken care of by the underlying architecture. We argue that ideal network connectivity is a service that any application should be able to use. It is a service that is composed from underlying (possibly orthogonal) capabilities of the node and the environment that the application then resides in. This obvious statement enables us to look at architecture from a service oriented point to view. This high level abstraction for a network has various advantages. Such and approach inherently accommodates the business boundaries existing in the real world networks such as commercial boundaries, administrative domains etc and helps traversing across these a natural part of service negotiation and usage. While this is also the approach of overlay networks, their implementation is not uniform or generic enough to apply to the network architecture as a whole. We propose to integrate the available communication services and current higher layer services in a unified and standardised manner.

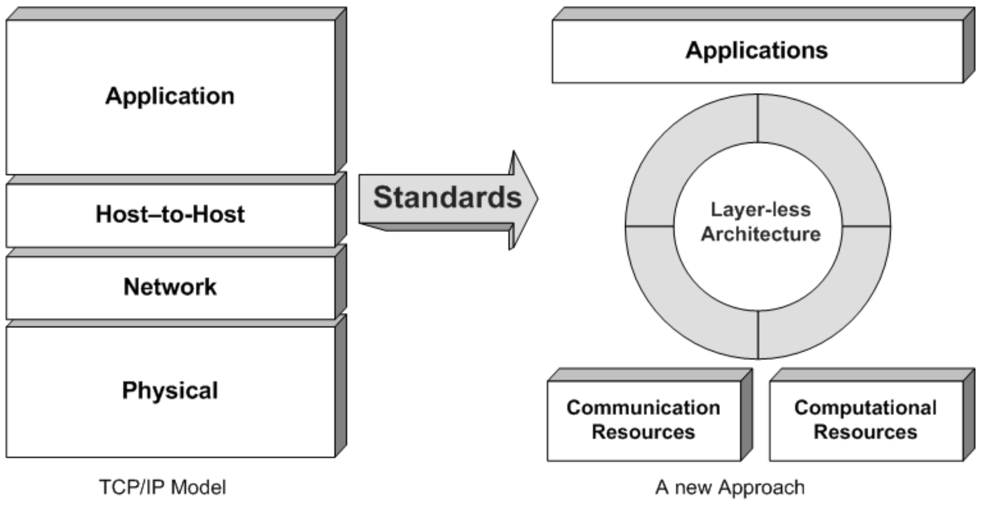


FIGURE A.2: The vision of *Network as a Service*.

A.6.2 Network Service as a Collection of Services

The next step is to identify the basis with which we compartmentalize the capabilities into modules, which can be later orchestrated. We take cues from the 'tussle' [7] being played out in the networking world to impose boundaries on the modules. A module encompasses a well defined service small enough to be a factor in the tussle but large enough to provide a specific, well defined, usable and non-trivial service. Identifying a set of modules which provides the services needed to compose a network architecture is not straight forward considering the fact that as a 'future' network architecture, it should be able to gracefully accommodate a wide spectrum of potential uses. The borders of modules and which attributes to consider as a module is of course, an open question. Different approaches can propose different modules to accomplish the same goals and it might be probable that it is impossible to agree on a standard set. But, as long as different modules provide a uniform interface and a clear description of its capabilities, the architectural principles we propose hold true.

From the above research directions, a sense of future direction can be derived. We can see not one 'Internet' but many 'Internets' of varying capabilities and characteristics (inter-network of things, inter-network of specific applications, inter-network of specific intentions etc.). This concept is already starting to evolve if we consider Peer-to-peer application 'networks' using the existing infrastructure as a transport network. This brings us very close to a similar problem that existed in enterprises towards the beginning of the 21st century. Enterprises built their IT systems to streamline their processes. Large distributed enterprises built middleware to support transactions and interconnect their systems across domains. Since there were no standards for such systems, these proprietary systems posed a problem during instances of interoperability (mergers, acquisitions, collaborations etc). The concept of Service Oriented Architecture (SOA) [97] was adopted to enable a standardised and open way for enterprises to open up their IT infrastructure for collaboration. We note the best known implementation of SOA in Web Services (WS-*) [94] specifications, standardization and its implementations. While, Web Services specifications specifically address Enterprise Application Integration (EAI [95]), we need a similar principle and framework to be applied to future heterogenous communication networks in order to interconnect business borders and administrative domains.

The simple sounding business goal of *connecting to customers, suppliers or partners electronically* [122] translated into web services that offer standard based mechanisms to create or compose services from composite and often cross-organizational components and Web based services [123]. We look at the underlying network architecture under the same requirement considerations, i.e, a service to be offered to applications, composed of various other services, local or distributed.

A.6.3 The Principle of Service Orientation

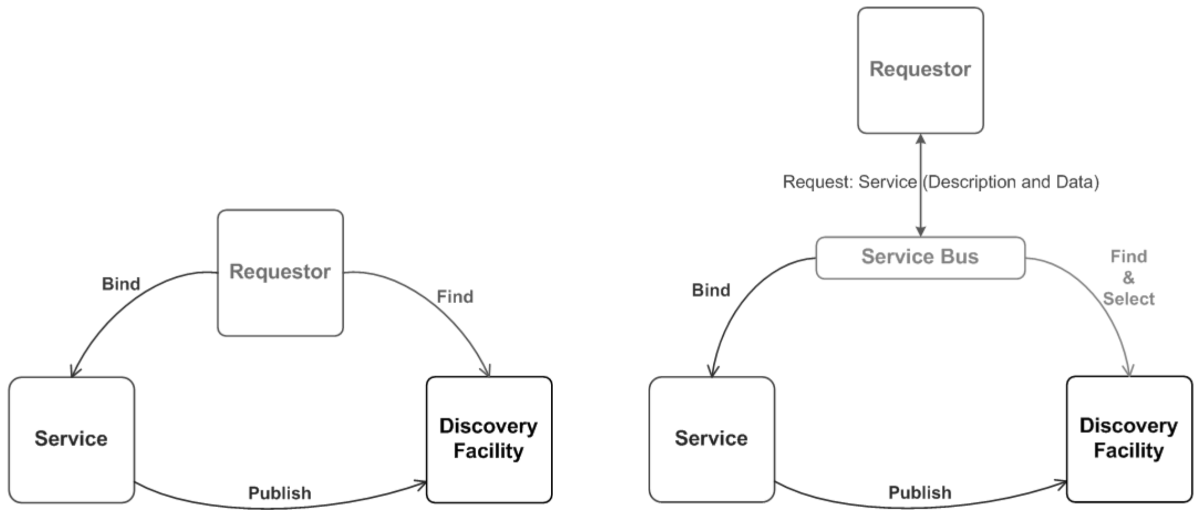
Before we apply service orientation principles to the network architecture, it is helpful to identity the definition of a *service*. A service as specific functionality is characterised by the following features [86]:

***Composable*:** An entity can use the service depending on the conditions specified either directly or as a part of another service.

***Describable*:** A service can be fully described including what functions it provides and how these functions can be accessed (e.g. through metadata).

***Discoverable*:** A service can be discovered based on their metadata by other services or entities.

SOA represents an abstract architectural concept of building software systems that is based on loosely coupled components (services) that have been described in a uniform way and can be discovered and composed [86]. The services that form the part of an SOA should be dynamically composable by any entity interested in availing it. The core elements that comprise an SOA is illustrated in Fig. A.3(a). This architecture is further extended with a Service Bus (SB) to make the discovery and binding process more transparent to the requesting service by visualizing the candidate services from the requestor's perspective (Fig. A.3(b)). The concept of SB is important as it forms a decision making entity or a middleware on behalf of the requesting service.



(a) SOA Triangle.

(b) SOA Triangle enhanced with Service Bus.

FIGURE A.3: Basic components of SOA architecture

We adopt this updated SOA approach to construct the elements for a Network Architecture. The abstraction of network as a service or a collection of services can be gracefully accommodated into this model.

A.7 A Relationship based Service Oriented Architecture

To propose a network architecture within the structure of SOA, it is necessary to define a framework or a collection of services and define the environment within which it functions. This is not a straight forward exercise, given that our intention is to define a generic architecture for the next generation communication networks. To arrive at such a architecture, we inspect the highest level abstraction required for any communication. When two entities meet (have access to a channel with another entity) and wish to communicate with each other. The *entity* here do not particularly refer to a node, a device, a piece of software or a service but could be any of the them. We would like to be agnostic here regarding where or what implements the intelligence for communication. We visualize the wish to communicate of two entities (with the intent and ability to do so) as the meeting of two *domains*.

Domains can be conceived as a region (physical or virtual) characterized by a specific feature and restricted by boundaries. A generic example for such a domain could be a business with business boundaries and characterised by business processes. This gives us a generic enough platform where services can reside and interact. For example, a transaction between services of different business entities (B2B) naturally classifies into a meeting of business domains. A physical device forms a domain with the common characteristic of a physical boundary and (usually) a single ownership. In this case as well, connecting two devices together can be reduced to a meeting of two domains. This *Concept of Domains* can be extended recursively for already collaborating domains, i.e, when two domains meet and agree upon the various aspects of service sharing and usage, their collaboration can be again abstracted as a domain, which being the collection of capabilities that they together can perform. The implications of such recursive nature derives from the composability of services within an SOA.

In this paper, We define 'Relation' as an association among dynamically collaborating nodes, devices and services in a network, characterized by a 'relationship metrics'. We propose a frame work termed 'Relationship Oriented Service Architecture'

(ROSA)[172] to agree upon a broad vocabulary that will be used to model recurring themes in ICT and integration environments. The aim of such a framework is to be able to reference one or more open specifications or standards for each identified service, that can be used to implement various version of the service. This is a flexibility available in the SOA. SOA principles are considered in all aspects of design including interfaces between modules and the relationship description. This enables us to decouple certain aspects or modules and develop it independently. Besides, this also implies that the services provided by some of the modules can be accomplished by a web based service or remote entity. This would be helpful in abstracting the network architecture across domains and networks without too much reworking of the communication network. This should also simplify integration with existing infrastructure and third party service providers (like standard based trust and security service providers).

The overall architecture provides an ‘Intelligent Middleware’ managing the communication, while providing an API towards the applications and managing connectivity resources independently. In SOA, the actual services are usually relatively simple ‘black boxes’ that can be applied in a flexible fashion in a variety of instances, as such avoiding duplication of functionality. While it is not efficient to dictate that all applications orchestrate and construct their own solutions from the above services, any large scale application can do so via the APIs that these services expose. For normal scenarios, the separation of application logic from transport logic to make applications communication agnostic can achieved by providing a higher level aggregate of these services via more specific generic APIs. This is done via the ‘Relationship Manager’ (RM). We focus on the argument that the given modules can be composed into a coherent architecture via a single paradigm, namely Relationships. We define ‘relation’ as an association among dynamically collaborating nodes, devices and services in a network. A relationship description contains parameters (‘relationship metrics’) to express the nature and background of the collaboration.

A.7.1 Relationship Manager

The relationship manager is itself a service which composes available services and offers APIs to accomplish most of the common services that applications need. The RM (a simplified version of 'Enterprise Bus' in Web Services) fits the numerous service components into a logical process (Orchestration) and facilitates the translation of data flow between services that may interpret a term differently. RM helps to avoid the 'monolithic silos' traditionally formed when implementing a complete usable service. The RM orchestrates the services for processes (on behalf of applications), based on contexts. Entities always communicate with services in their environment according to a certain context. Orchestrating these services *in context* provides the definition of relationships and causalities between different services of an entity communication space [175]. The context is expressed in the context service and the RM contains the logic to use it for orchestration.

By being a service by itself, the RM is replaceable with other implementations offering similar interfaces. However, to be truly modular and avoid tussle in the core, the relationship manager should be minimal, analogous to a micro-kernel in an operating system. This means that all useful services should be implemented outside the RM, or in services space and the only service that the RM provides is a meaningful orchestration. However, even for environments where the applications directly manage the orchestration, RM must be present for bootstrapping purposes.

From a different perspective, the RM virtualizes the available services to the applications. This is similar to the concept in Fig. A.3(b) where the Service Bus virtualizes the candidate services from the requestor's point of view. The RM can be thought of as a local instantiation of a Service Bus, which simplifies the search, query, binding and access of other services within the ROSA framework.

A.7.2 Service Composition

Fig. A.4 indicates an example of how a network can be composed out of available services. The services indicated are highly aggregated services from other elementary services available. The simple file transfer scenario mentioned above can be addressed by encapsulating the available network interfaces (Wi-Fi, bluetooth, Wireless USB, NFC etc.) and related protocols via the 'connectivity' service. This approach detaches the applications from having to be aware of the device location, neighboring nodes and their capabilities. If both devices support such a framework, then the underlying services can be combined by the relationship manager to accomplish a request similar to 'transfer the file to the closest neighbor after verification, via the lowest cost path'. The application need not know whether the interface used is Wi-Fi or bluetooth or NFC, or even how the phone discovered its neighbor. The detachment of network services also enable applica-

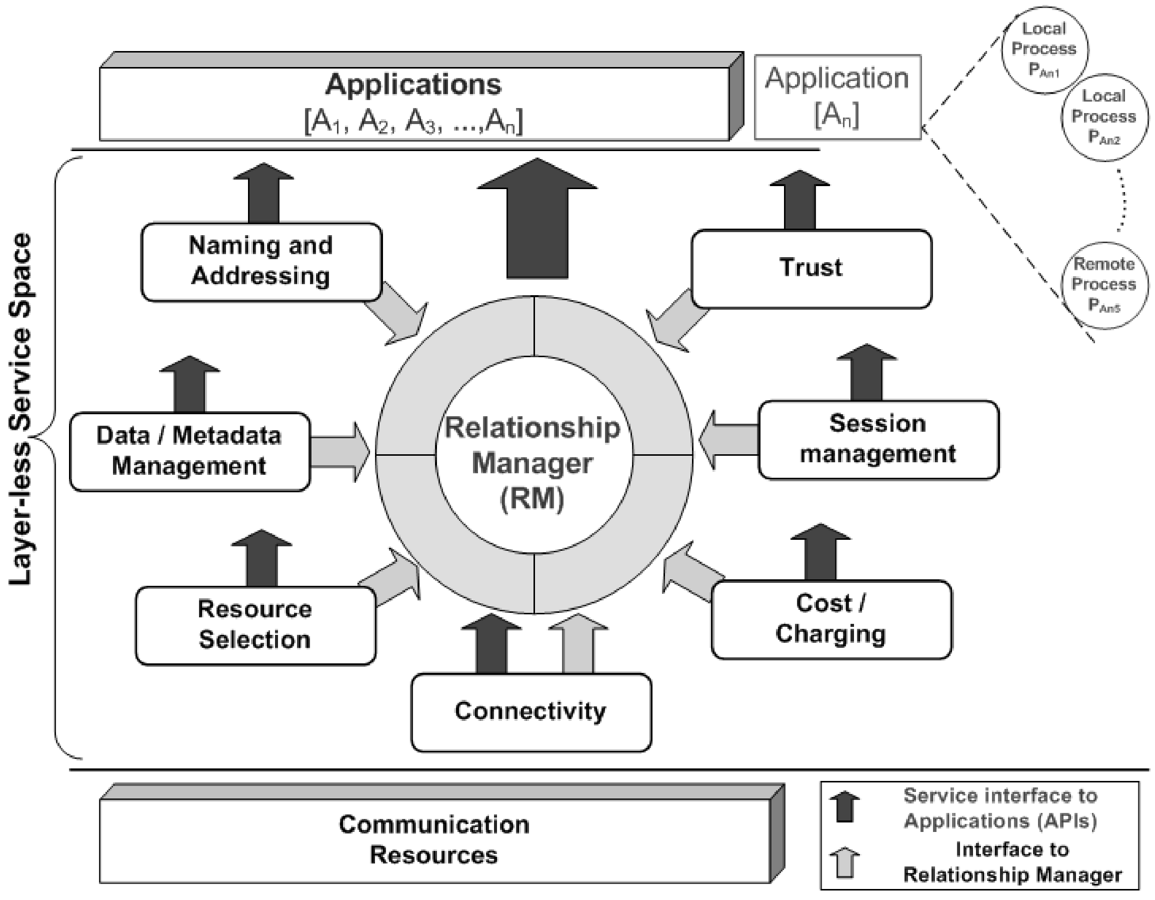


FIGURE A.4: A sample ROSA instance

tions to fully explore the ‘late binding’ (addressing the entity but postponing the entity location till the last possible instance) or ‘relationship composition’ by the relationship manager, as it can wait as late as possible to compose the architecture, utilizing the latest context/network/location updates and status.

A.7.3 Services

A selection of ‘building block’ services, which can be orchestrated to form higher level services is depicted in Fig. A.5 to visualize the framework. The services are clustered into logical groups to aid readability and no dependencies or explicit associations exist between service definitions. In practice, if several services with similar capabilities are exposed in an environment, the service interface may be realized using a shared implementation. For example, presence, context and messaging could all be managed using a single Jabber service. Or, authentication, authorization and accounting/charging (AAA) can be done via a single AAA service. A brief intentions of each services are mentioned in [172]. This is by no means an exhaustive list of services that can be made available, as the service oriented approach facilitates adding more or deducting unused services according to requirements. The core task of creating such a framework is to identify a broad set of services that need to be defined (called ‘factoring services’). We consider factoring to be an ongoing process as experience informs the choice of services, identifies shortcomings and indicate the services that require creation, discount, splitting or joining. A service is a pattern that can used to solve a specific problem and can be defined with in a framework at different levels [125]. While this pattern can be defined at various degrees of granularity [86], we propose them as a definition of function and scope. The functional focus provides the capacity to be specific about the range of expected behaviour of individual service, while being agnostic with regards to the implementation details or design of solutions. While this definition is far from sufficient to implement a network architecture, it provides as a starting grid to more detailed specifications and a reference model. From the abstract models of services, it is possible to derive XML

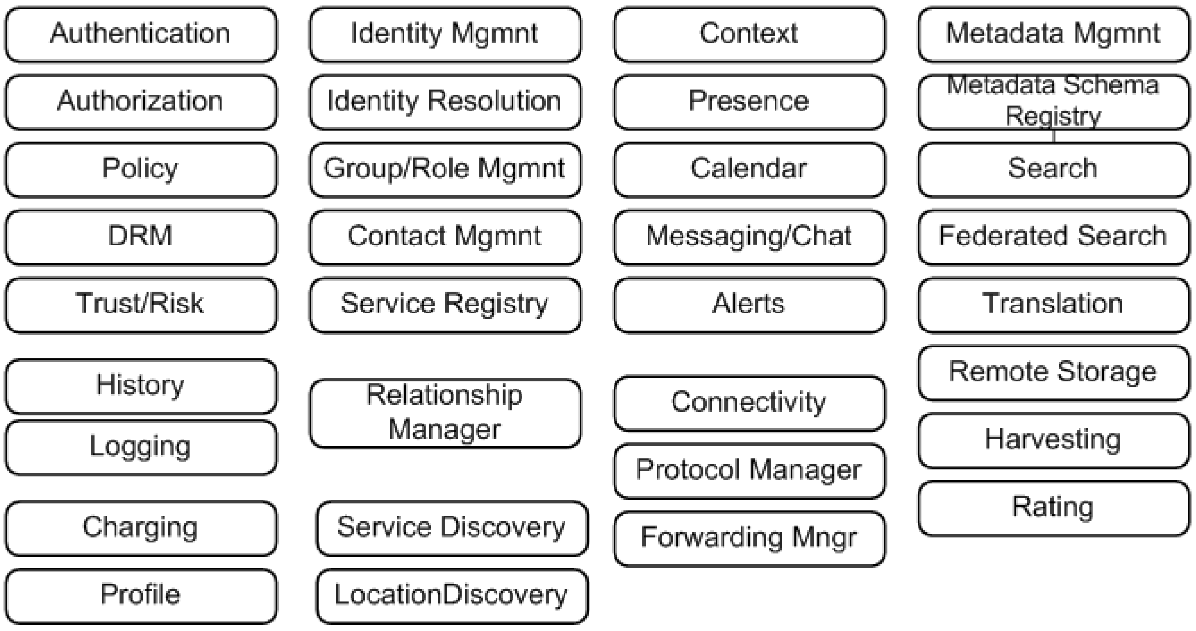


FIGURE A.5: ROSA Framework Services

schemas that define data to be exchanged. This approach enables specific SOA standard based definition for services to be implemented (as a Web-Service, for example). Such a framework is realized in an application with an interface to access a service that has commonly agreed operation definitions (e.g. Web Service Definition language, WSDL) and data structures (e.g XML schemas) [98].

There are different methods to realize SOA in communication systems. Web-Services represent one important approach and is the most adopted and widespread within the IT industry. There are various other Middleware (OMG CORBA, MSDCOM etc.) that can be used for such abstraction, but Web-Services have marked advantages like being much more loose coupled, dynamic and adaptable to change. Besides, it supports and open way to develop specifications and using technology via a broad consortia, which takes into account the stakeholder interests.

The world wide consortium (W3C) defines Web-Services as:

A Web-Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in *a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using Simple Object Access Protocol (SOAP) messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

Web-Services provide a uniform way of describing components or services with in a network, locating them and accessing them. Web-Service specifications define formats and protocols that allow services to interoperate across those vendor platforms that provide conforming implementations, either natively or by mapping them onto existing proprietary middleware offerings. The standards and specifications that are adopted are being developed in an open way through community organizations like W3C and OASIS. The process allows for a consensus based standardization and vetting by commercial interests before being accepted or approved as a standard. Web services rely on XML for the basic underlying data model, SOAP for the message processing and data model, and WS-Addressing for addressing services and identifying messages independent of transport.

SOAP is a simple and extensible XML based communication protocol that provides a way to communicate between applications running on different operating systems, with different technologies and programming languages. With SOAP, the underlying transport might change as the message is routed between nodes, even the mechanism selected for each hop may vary as required. An important facility is the feature that the messages can be routed based on the content of the headers and the data inside the message body. SOAP forms the messaging framework of ROSA, owing to these attributes. Another specification, WS-Addressing provides an interoperable transport independent way for identifying message senders and receivers associated with a message exchange. Besides securing end-to-end endpoint identification in messages, this specification enables messaging systems to support message transmission the networks that include middleboxes like endpoint managers, firewalls, gateways etc. Further details of these specifications are available at [94, 97].

The above mentioned services can take advantage of the Service Descriptions framework available in WS which defines metadata that fully describes the characteristics of a service that are deployed in the network. This metadata provides the abstract definition of the information necessary to deploy and interact with a service. In Web-Services, Web Services Description language (WSDL) is most widely used for describing metadata for Web-Services, i.e, what a Service can do. It offers a standard language-agnostic view of services offered by a Web-Service. WSDL is an XML format for describing services as a set of endpoints that operate on messages containing either document-oriented or procedure oriented information and is well suited for the ROSA framework.

A Policy service can use the WS-Policy [102], an extensible framework for Web-Services constraints and conditions allowing for a clear and and uniform expression of of the available options. The *Service Discovery* module can query over metadata that describes services. This metadata should be searchable and discoverable for users to find and use services. Hence, aggregation and discovery services for metadata which in turn becomes a repository or registry for services are very useful, if not unavoidable. Universal Description Discovery and Integration (UDDI) is the most common used specification for a Web-Service registry.

As can be realized at this point, the Service Oriented Architectural approach for Networking through the ROSA framework can be realized using the various Web-Services specifications (WS-*). The next step is to select a subset of the services and create a basic profile, which can bootstrap into a simple functioning model using Web-Services. This is a future exercise related to this proposal.

A.7.4 Comparison with other Approaches

A clear study of the drawbacks of such an approach needs to be carried out, to propose improvements to the architecture at initial stages. Coupling of numerous independent services which communicate among themselves locally or over a network will generate a large quantity of messages. The quantity of messages increases significantly with the number of services utilized indicating scalability issues. This might impose a limit on the modularity of services, as smaller services imply greater flexibility but generate higher overhead traffic. We can address this complexity partly through the use of RM which becomes the only service that each of the other services interact with, thus reducing inter-service messaging. The cost, however, is a significant complexity in the RM to correctly fit the numerous services into a logical solution for different contexts.

Efficiency for small set of services compared to legacy architectures will be less, using our approach. However, the advantages of SOA will be inherited into the proposed architecture. The integration of components within heterogenous environments or dynamically changing component configurations is best addressed using our architecture. SOA and Web-Services offer potentially significant benefits to large service sets that undergo frequent change and facilitate reusability. Currently, the XML footprint and parsing cost at both ends of a message exchange does take up time and resources. With high performance as the criterion, Web-Services might not be as efficient as current architectures. Binary XML for interchange can improve the performance, but it is yet to be standardized.

A.8 Conclusions and Future Work

For a 5th generation network, the principles and paradigms extend beyond bandwidth or ubiquitous computing. In such a network, every device and service is immersed in an environment of usable services, transparent to the location, be it local or remote. Network communication becomes an enabler or a pure service, where the focus shift from connectivity to usage scenarios. Processing power on demand, expandable storage, guaranteed service qualities etc will become the norm. For such a network, it is necessary that applications and users should not have to deal with low level infrastructure attributes, but an high level abstraction of such attributes as usable services. The architectural principles proposed in this article is a step towards such a networking vision.

We introduced a new architecture that treats network resources and service resources in the same way by applying service oriented principles. In addition to hiding heterogeneity and respecting business borders, this approach gives enough flexibility to migrate from today's internet towards a network of future *Internets*.

APPENDIX B

SELECTION OF PAPERS

This appendix lists the author's publications related to the work done towards the fulfilment of this thesis.

[1] R. Kumar,"*Fifth Generation Networking Principles for a Service Driven Future Internet Architecture*," Wireless Personal Communications, vol. 55, 2010.

[2] R. Kumar, A. Haber, A. Yazidi, and F. Reichert, "*Towards a relation oriented service architecture*" in COMSNETS'10: Proceedings of the 2nd international conference on COMmunication systems and NETworks, pp. 452-459, 2010.

[3] R. Kumar, F. Reichert, A. Haber, A. Aasgard, and Lian Wu, "*Migration of Enterprise VoIP/SIP Solutions towards IMS*", in Testbeds and Research Infrastructure for the Development of Networks and Communities, 2007. TridentCom 2007. 3rd International Conference on, pp. 1-6, 2007.

[4] A. Häber, M. Gerdes, F. Reichert, A. Fasbender, and R. Kumar, "*Delivering Services to Residential Appliances by Utilizing Remote Resource Awareness*," in Proceedings of

the 2008 The Second International Conference on Next Generation Mobile Applications, Services, and Technologies, pp. 161-166, 2008.

[5] A. Haber, M. Gerdes, F. Reichert, A. Fasbender, and R. Kumar, "*Remote Service Usage Through Sip with Multimedia Access as a Use Case*," in Personal, Indoor and Mobile Radio Communications, 2007. PIMRC 2007. IEEE 18th International Symposium on, pp. 1-5, 2007.

REFERENCES

[1] ITU-T, "Itu-t world telecommunication/ict indicators database," Online, 2010. [Online]. Available: http://www.itu.int/ITU-D/ict/statistics/

[2] M. Conti, G. Maselli, G. Turi, and S. Giordano, "Cross-layering in mobile ad hoc network design," *Computer*, vol. 37, no. 2, pp. 48–51, 2004.

[3] M. Gregg and S. Watkins, *Hack the Stack: Using Snort and Ethereal to Master the 8 Layers of an Insecure Network*. Syngress Publishing, 2006.

[4] T. Erl, A. Karmarkar, P. Walmsley, H. Haas, L. U. Yalcinalp, K. Liu, D. Orchard, A. Tost, and J. Pasley, *Web Service Contract Design and Versioning for SOA*, 1st ed. Prentice Hall, Oct. 2008.

[5] T. Erl, *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007.

[6] V. G. Cerf, Robert, and E. Icahn, "A protocol for packet network intercommunication," *IEEE Trans*, p. 637, 1974.

[7] D. Clark, J. Wroclawski, K. Sollins, and R. Braden, "Tussle in cyberspace: defining tomorrow's internet," *Networking, IEEE/ACM Transactions on*, vol. 13, no. 3, pp. 462–475, 2005.

[8] R. H. Zakon. Hobbe's internet timeline. [Online]. Available: http://www.zakon.org/robert/internet/timeline/

[9] M. Handley, "Why the internet only just works," *BT Technology Journal*, vol. 24, no. 3, pp. 119–129, 2006.

[10] M. S. Blumenthal and D. D. Clark, "Rethinking the design of the internet: the end-to-end arguments vs. the brave new world," *ACM Trans. Interet Technol.*, vol. 1, no. 1, pp. 70–109, 2001.

[11] B. Carpenter and K. Moore, "Connection of ipv6 domains via ipv4 clouds," RFC 3056 (Proposed Standard), Internet Engineering Task Force, Feb. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3056.txt

[12] J. Stine, "Cross-layer design of manets: The only option," in *Military Communications Conference, 2006. MILCOM 2006. IEEE*, 2006, pp. 1–7.

[13] A. Goldsmith and S. Wicker, "Design challenges for energy-constrained ad hoc wireless networks," *Wireless Communications, IEEE*, vol. 9, no. 4, pp. 8–27, 2002.

[14] N. Kshetri, "Pattern of global cyber war and crime: A conceptual framework," *Journal of International Management*, vol. 11, no. 4, pp. 541–562, December 2005. [Online]. Available: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=842265

[15] K. J. Knapp and W. R. Boulton, "Cyber-warfare threatens corporations: Expansion into commercial environments," *Information Systems Management*, vol. 23, no. 2, p. 76, 2006. [Online]. Available: http://www.informaworld.com/10.1201/1078.10580530/45925.23.2.20060301/92675.8

[16] T. Porter. The perils of deep packet inspection. http://www.securityfocus.com/infocus/1817. [Online]. Available: http://www.securityfocus.com/infocus/1817

[17] W. Eddy, "At what layer does mobility belong?" *Communications Magazine, IEEE*, vol. 42, no. 10, pp. 155–159, 2004.

[18] M. Ratola, "Which layer for mobility? - comparing mobile ipv6, hip and sctp," *HUT T-110.551 Seminar on Internetworking*, 2004.

[19] A. Weiss, "Computing in the clouds," *netWorker*, vol. 11, no. 4, pp. 16–25, 2007. [Online]. Available: http://portal.acm.org/ft_gateway.cfm?id=1327513&type=html&coll=GUIDE&dl=GUIDE&CFID=97904226&CFTOKEN=85754888

[20] M. Vouk, "Cloud computing - issues, research and implementations," in *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, 2008, pp. 31–40. [Online]. Available: 10.1109/ITI.2008.4588381

[21] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, 2008, pp. 5–13. [Online]. Available: 10.1109/HPCC.2008.172

[22] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1496100

[23] L. Youseff, M. Butrico, and D. D. Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop, 2008. GCE '08*, 2008, pp. 1–10. [Online]. Available: 10.1109/GCE.2008.4738443

[24] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, S. Tuecke, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems," in *Job Scheduling Strategies for Parallel Processing*. Springer Berlin Heidelberg, Nov. 2002, vol. 2537, pp. 153–183. [Online]. Available: http://dx.doi.org/10.1007/3-540-36180-4_9

[25] I. Brandic, D. Music, and S. Dustdar, "Service mediation and negotiation bootstrapping as first achievements towards self-adaptable grid and cloud services," in *Proceedings of the 6th international conference industry session on Grids meets autonomic computing*. Barcelona, Spain: ACM, 2009, pp. 1–8. [Online]. Available: http://portal.acm.org/citation.cfm?id=1555302

[26] V. Srivastava and M. Motani, "Cross-layer design: a survey and the road ahead," *Communications Magazine, IEEE*, vol. 43, no. 12, pp. 112 – 119, 12 2005.

[27] I. Akyildiz, M. Vuran, and O. Akan, "A cross-layer protocol for wireless sensor networks," 03 2006, pp. 1102 –1107.

[28] P. Mahonen, J. Riihijarvi, M. Petrova, and Z. Shelby, "Hop-by-hop toward future mobile broadband ip," *Communications Magazine, IEEE*, vol. 42, no. 3, pp. 138–146, 2004.

[29] I. G. Niemegeers and S. M. H. D. Groot, "From personal area networks to personal networks: A user oriented approach," *Wirel. Pers. Commun.*, vol. 22, no. 2, pp. 175–186, 2002.

[30] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish, "A layered naming architecture for the internet," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*. Portland, Oregon, USA: ACM, 2004, pp. 343–352.

[31] D. R. Cheriton and M. Gritter. (2000) Triad: A scalable deployable nat-based internet architecture. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.4093

[32] P. Nikander, "The host identity protocol (hip): Bringing mobility, multi- homing, and baseline security together," in *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, 2007, pp. 518–519.

[33] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE COMMUNICATIONS SURVEYS AND TUTORIALS*, vol. 7, pp. 72—93, 2005.

[34] M. Samad, F. Yusuf, H. Hashim, M. Mahfudz, and M. Zan, "Deploying internet protocol version 6 (ipv6) over internet protocol version 4 (ipv4) tunnel," in *Research and Development, 2002. SCOReD 2002. Student Conference on*, 2002, pp. 109–112.

[35] B. Ahlgren, L. Eggert, B. Ohlman, and A. Schieder, "Ambient networks: bridging heterogeneous network domains," in *Proc. 16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, vol. 2, 2005, pp. 937–941 Vol. 2.

[36] J. Scott, J. Crowcroft, P. Hui, and C. Diot, "Haggle: a networking architecture designed around mobile users," in *Proceedings of the Third Annual Conference on Wireless On-demand Network Systems and Services*, 2006, pp. 86, 78.

[37] D. D. Clark and K. Sollins, "Addressing reality: An architectural response to demands on the evolving internet," *In ACM SIGCOMM Workshop on Future Directions in Network Architecture*, pp. 247—257, 2003.

[38] R. Braden, T. Faber, and M. Handley, "From protocol stack to protocol heap: role-based architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 17–22, 2003.

[39] B. Wong, A. Slivkins, and E. Sirer, “Meridian: A lightweight network location service without virtual coordinates,” *IN SIGCOMM*, pp. 85—96, 2005.

[40] I. Castineyra, N. Chiappa, and M. Steenstrup, “The nimrod routing architecture,” RFC 1992, Internet Engineering Task Force, August 1996. [Online]. Available: http://www.ietf.org/rfc/rfc1992.txt

[41] D. Clark, R. Braden, A. Falk, and V. Pingali, “Fara: reorganizing the addressing architecture,” in *FDNA ’03: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*. New York, NY, USA: ACM, 2003, pp. 313–321.

[42] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, “A survey of active network research,” *IEEE COMMUNICATIONS MAGAZINE*, vol. 35, pp. 80—86, 1997.

[43] D. Clark, “The design philosophy of the darpa internet protocols,” in *SIGCOMM ’88: Symposium proceedings on Communications architectures and protocols*. New York, NY, USA: ACM, 1988, pp. 106–114.

[44] R. L. Aguiar, “Some comments on hourglasses,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 5, pp. 69–72, 2008.

[45] R. Atkinson, S. Bhatti, and S. Hailes, “A proposal for unifying mobility with multi-homing, nat & security,” in *MobiWac ’07: Proceedings of the 5th ACM international workshop on Mobility management and wireless access*. New York, NY, USA: ACM, 2007, pp. 74–83.

[46] V. Srivastava and M. Motani, “Cross-layer design: a survey and the road ahead,” *Communications Magazine, IEEE*, vol. 43, no. 12, pp. 112–119, 2005.

[47] S.-H. Choi, D. Perry, and S. Nettles, “A software architecture for cross-layer wireless network adaptations,” in *Software Architecture, 2008. WICSA 2008. Seventh Working IEEE/IFIP Conference on*, 18-21 2008, pp. 281 –284.

[48] J. Burbank and W. Kasch, “Cross-layer design for military networks,” in *Military Communications Conference, 2005. MILCOM 2005. IEEE*, 2005, pp. 1912–1918 Vol. 3.

[49] V. Kawadia and P. Kumar, “A cautionary perspective on cross-layer design,” *Wireless Communications, IEEE*, vol. 12, no. 1, pp. 3 – 11, feb. 2005.

[50] E. Lear and R. Droms, *What’s In A Name: Thoughts from the NSRG*, Internet Research Task Force (IRTF) Internet Draft, 2004, name Space Research Group (NSRG). [Online]. Available: http://tools.ietf.org/html/draft-irtf-nsrg-report-10

[51] R. Moskowitz and P. Nikander, “Host identity protocol (hip) architecture,” RFC 4423 (Informational), Internet Engineering Task Force, May 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4423.txt

[52] E. Rescorla, “Diffie-hellman key agreement method,” RFC 2631 (Proposed Standard), Internet Engineering Task Force, Jun. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2631.txt

[53] S. Kent, “Ip encapsulating security payload (esp),” RFC 4303 (Proposed Standard), Internet Engineering Task Force, Dec. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4303.txt

[54] M. O’Dell, “8+8 - an alternate addressing architecture for ipv6,” 10 1996.

[55] E. Nordmark and M. Bagnulo, “Shim6: Level 3 multihoming shim protocol for ipv6,” June 2009. [Online]. Available: http://tools.ietf.org/html/rfc5533

[56] N. Niebert, A. Schieder, J. Zander, and R. Hancock, *Ambient Networks: Cooperative Mobile Networking for the Wireless World.* Wiley Publishing, 2007.

[57] European fp6 project, european commission. [Online]. Available: http://europa.eu.int/comm/research/fp6/index_en.html

[58] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, “The design and implementation of an intentional naming system,” in *SOSP ’99: Proceedings of the seventeenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 1999, pp. 186–201.

[59] M. Gudgin, M. Hadley, and T. Rogers, *Web Services Addressing 1.0 - Core*, World Wide Web Consortium W3C Recommendation, May 2006. [Online]. Available: http://www.w3.org/TR/ws-addr-core/

[60] A. Wong and A. Yeung, *Network Infrastructure Security*. Springer Science Business Media LLC, 2009, vol. 1, no. 978-1-4419-0165-1.

[61] R. Hinden and S. Deering, “Internet protocol version 6 (ipv6) addressing architecture,” April 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3513.txt

[62] D. Harkins and D. Carrel, “The internet key exchange (ike),” November 1998. [Online]. Available: http://tools.ietf.org/html/rfc2409

[63] C. Perkins, “Mobile ip,” *Communications Magazine, IEEE*, vol. 40, no. 5, pp. 66 –82, may. 2002.

[64] D. Johnson, C. Perkins, and J. Arkko, “Mobility support in ipv6,” Network Working Group, June 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3775.txt

[65] A. Ford, P. Eardley, and B. van Schewick, “New design principles for the internet,” in *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*, June 2009, pp. 1–5.

[66] T. Koponen and A. Gurtov, “Application mobility with host identity protocol,” *In Proc. of NDSS Wireless and Security Workshop*, 2005.

[67] S. Novaczki, L. Bokor, and S. Imre, “A hip based network mobility protocol,” in *SAINT-W ’07: Proceedings of the 2007 International Symposium on Applications*

and the Internet Workshops. Washington, DC, USA: IEEE Computer Society, 2007, p. 48.

[68] P. Nikander, J. Ylitalo, and J. Wall, “Integrating security, mobility, and multihoming in a hip way,” in *NDSS’03: Proceedings of the Network and Distributed Systems Security Symposium*, Feb. 2003, pp. 87–99.

[69] Y. Rekhter and T. Li, “An architecture for ip address allocation with cidr,” September 1993. [Online]. Available: http://tools.ietf.org/html/rfc1518

[70] B. Ahlgren, L. Eggert, B. Ohlman, J. Rajahalme, and A. Schieder, “Names, addresses and identities in ambient networks,” in *DIN ’05: Proceedings of the 1st ACM workshop on Dynamic interconnection of networks*. New York, NY, USA: ACM, 2005, pp. 33–37.

[71] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica, “Rofl: routing on flat labels,” *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 363–374, 2006.

[72] R. E. Schantz, J. P. Loyall, C. Rodrigues, D. C. Schmidt, Y. Krishnamurthy, and I. Pyarali, “Flexible and adaptive qos control for distributed real-time and embedded middleware,” in *Middleware ’03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2003, pp. 374–393.

[73] P. White, “Rsvp and integrated services in the internet: a tutorial,” *Communications Magazine, IEEE*, vol. 35, no. 5, pp. 100–106, 1997. [Online]. Available: 10.1109/35.592102

[74] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, “Rsvp: a new resource reservation protocol,” *Communications Magazine, IEEE*, vol. 40, no. 5, pp. 116 –127, may. 2002.

[75] J. Saarnio, R. Aguiar, and I. V. Kumar, “Layereless communications: From dream to reality?” *Wirel. Pers. Commun.*, vol. 44, no. 1, pp. 51–55, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1325281

[76] R. Prasad, “A perspective of layerless communications,” *Wireless Personal Communications*, vol. 44, no. 1, pp. 95–100, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11277-007-9385-x

[77] J. Su, J. Scott, P. Hui, J. Crowcroft, E. de Lara, C. Diot, A. Goel, M. Lim, and E. Upton, *Haggle: Seamless Networking for Mobile Applications.*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4717, pp. 391–408. [Online]. Available: http://www.cl.cam.ac.uk/~ph315/publications/haggle-ubicomp2007.pdf

[78] P. Hui, A. Chaintreau, R. Gass, J. Scott, J. Crowcroft, and C. Diot, “Pocket switched networking: Challenges, feasibility, and implementation issues,” in *Proceedings of the Workshop on Autonomic Communications*, 2005.

[79] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, “A data-oriented (and beyond) network architecture,” *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 181–192, 2007.

[80] H. Bannazadeh and A. Leon-Garcia, “On the emergence of an application-oriented network architecture,” in *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*. IEEE Computer Society, 2007, pp. 47–54.

[81] K. Knightson, N. Morita, and T. Towle, “Ngn architecture: generic principles, functional architecture, and implementation,” *Communications Magazine, IEEE*, vol. 43, no. 10, pp. 49–56, 2005.

[82] A. S. Tanenbaum and A. S. Woodhull, *Operating Systems Design and Implementation (3rd Edition) (Prentice Hall Software Series)*. Prentice Hall, January 2006.

[83] J. Dollimore, T. Kindberg, and G. Coulouris, *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science Series).* Addison Wesley, May 2005. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0321263545

[84] S. Vinoski, "Corba: integrating diverse applications within distributed heterogeneous environments," *Communications Magazine, IEEE*, vol. 35, no. 2, pp. 46 –55, feb. 1997.

[85] T. B. Downing, *Java RMI: Remote Method Invocation.* Foster City, CA, USA: IDG Books Worldwide, Inc., 1998.

[86] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. Ferguson, *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More.* Prentice Hall PTR, march 2005. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0131488740

[87] T. Erl, *Service-Oriented Architecture : A Field Guide to Integrating XML and Web Services.* Prentice Hall PTR, Apr. 2004. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0131428985

[88] J. Lee, K. Siau, and S. Hong, "Enterprise integration with erp and eai," *Commun. ACM*, vol. 46, no. 2, pp. 54 – 60, 2003.

[89] J. Rattner and G. Cox, "Object-based computer architecture," *SIGARCH Comput. Archit. News*, vol. 8, no. 6, pp. 4–11, 1980.

[90] A. Adi, S. Stoutenburg, and S. Tabet, Eds., *Rules and Rule Markup Languages for the Semantic Web.* Berlin/Heidelberg: Springer-Verlag, 2005, vol. 3791. [Online]. Available: http://www.springerlink.com/content/y58637312834r741/

[91] R. Baldoni, M. Contenti, and A. Virgillito, "The evolution of publish/subscribe communication systems," pp. 137–141, 2003.

[92] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, “Service-oriented computing: State of the art and research challenges,” *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[93] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services: Concepts, Architectures and Applications*. Berlin: Springer, 2004.

[94] W3C. W3c web services architecture. [Online]. Available: http://www.w3.org/TR/ws-arch/

[95] D. S. Linthicum, *Enterprise application integration*. Essex, UK, UK: Addison-Wesley Longman Ltd., 2000.

[96] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon, *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*, World Wide Web Consortium W3C Recommendation, April 2007. [Online]. Available: http://www.w3.org/TR/soap/

[97] OASIS. Oasis soa commitee. [Online]. Available: http://www.oasis-open.org/committees/tc_cat.php?cat=soa

[98] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, “Web service definition language (wsdl),” World Wide Web Consortium, Tech. Rep. NOTE-wsdl-20010315, March 2001. [Online]. Available: http://www.w3.org/TR/wsdl

[99] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, “Web services description language (wsdl) version 2.0 part 1: Core language,” June 2007. [Online]. Available: http://www.w3.org/TR/wsdl20/

[100] L. Clement, A. Hately, C. von Riegen, and T. Rogers, *OASIS Universal Description Discovery and Integration (UDDI) Specification*, Organization for the Advancement of Structured Information Standards (OASIS) UDDI Spec Technical Committee Draft 20 041 019, Rev. 3, February 2005. [Online]. Available: http://www.oasis-open.org/committees/uddi-spec

[101] U. Küster, H. Lausen, and B. König-Ries, "Evaluation of semantic service discovery - a survey and directions for future research," in *Proceedings of the 2nd Workshop on Emerging Web Services Technology (WEWST07) in conjunction with the 5th IEEE European Conference on Web Services (ECOWS07)*, vol. 2, 2008, pp. 41–58.

[102] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, and Ümit Yalçinalp, "Web services policy framework (wspolicy)," September 2007. [Online]. Available: http://www.w3.org/TR/ws-policy

[103] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, *Web Services Security (WS-Security)*, OASIS OASIS Standard Specification, Rev. 1.1, February 2006. [Online]. Available: http://docs.oasis-open.org/wss/v1.1

[104] I. Robinson and P. Knight, *OASIS Web Services Transaction (WS-TX)*, OASIS OASIS Standard, Rev. 1.2, February 2009. [Online]. Available: www.oasis-open.org/committees/ws-tx

[105] Oasis web services reliable messaging (wsrm). [Online]. Available: http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf

[106] R. Monson-Haefel, *J2EE(TM) Web Services*, 1st ed. Addison-Wesley Professional, 10 2003.

[107] M. Stal, "Web services: beyond component-based computing," *Commun. ACM*, vol. 45, no. 10, pp. 71–76, 2002. [Online]. Available: http://portal.acm.org/ft_gateway.cfm?id=570934&type=html&coll=GUIDE&dl=GUIDE&CFID=62266538&CFTOKEN=22342330

[108] R. Chumbley, J. Durand, G. Pilz, and T. Rutt, *WS-Interoperability (WS-I) Basic Profile Version 2.0*, Web Services interoperability Organization Working Group Draft 20 100 331, Rev. 2, March 2010. [Online]. Available: http://ws-i.org/profiles/BasicProfile-2.0-WGD.html

[109] W. Vogels, “Web services are not distributed objects,” *IEEE Internet Computing*, vol. 7, no. 6, pp. 59–66, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?id=1050716

[110] A. Sheth and J. A. Miller, “Web services: Technical evolution yet practical revolution,” *IEEE Intelligent Systems (IEEEIS)*, vol. 18, pp. 78–80, 2003.

[111] R. Kazman, G. Abowd, L. Bass, and P. Clements, “Scenario-based analysis of software architecture,” *IEEE Software*, vol. 13, no. 6, pp. 47–55, 1996.

[112] R. Kazman, L. Bass, G. Abowd, and M. Webb, “Saam: a method for analyzing the properties of software architectures,” in *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on*, 1994, pp. 81–90.

[113] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, Second Edition*. Addison-Wesley Professional, Apr. 2003. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0321154959

[114] S. Lohr, “The web’s inventor regrets one small thing,” *The New York Times*, Oct. 2009. [Online]. Available: http://bits.blogs.nytimes.com/2009/10/12/the-webs-inventor-regrets-one-small-thing/

[115] R. Kazman, R. Nord, and M. H. Klein, “A life-cycle view of architecture analysis and design methods,” Carnegie Mellon, Technical Note CMU/SEI-2003-TN-026, Sep. 2003. [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/03tn026.cfm

[116] M. R. Barbacci, R. J. Ellison, A. J. Lattanze, J. A. Stafford, C. B. Weinstock, and W. G. Wood, “Quality attribute workshops qaws - third edition,” Carnegie Mellon, Technical Report CMU/SEI-2003-TR-016, Oct. 2003. [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/03tr016.cfm

[117] R. Kazman, J. Asundi, and M. Klein, "Quantifying the costs and benefits of architectural decisions," in *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on*, 2001, pp. 297–306.

[118] P. C. Clements, "Active reviews for intermediate designs," Carnegie Mellon, Technical Note CMU/SEI-2000-TN-009, Aug. 2000. [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/00tn009.cfm

[119] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," in *Engineering of Complex Computer Systems, 1998. ICECCS '98. Proceedings. Fourth IEEE International Conference on*, 1998, pp. 68–78.

[120] C. Larman and V. R. Basili, "Iterative and incremental development: A brief history," *Computer*, vol. 36, no. 6, pp. 47–56, 2003.

[121] S. Wilson, K. Blinco, and D. Rehak, "Service-oriented frameworks: Modelling the infrastructure for the next generation of e-learning systems," Altilab, Tech. Rep. 1, July 2004, a Paper prepared on behalf of DEST (Australia), JISC-CETIS (UK), and Industry Canada.

[122] C. S. Corporation, "13th ann. critical issues of is management survey," Computer Sciences Corporation, Tech. Rep., 2000. [Online]. Available: www.csc.com/survey

[123] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, 2003.

[124] D. A. Chappell, *Enterprise service bus*. O'Reilly Media, Inc., Jun. 2004.

[125] P. Nicholls, "Enterprise architectures and the international e-framework," e-Framework Partnership for Education and Research, Tech. Rep. 1.3 Final, Jul. 2009. [Online]. Available: http://www.e-framework.org/Portals/9/docs/EAPaper_2009-07.pdf

[126] F. A. Rabhi and S. Gorlatch, Eds., *Patterns and skeletons for parallel and distributed computing*. London, UK: Springer-Verlag, 2003.

[127] Y. Kim and K.-G. Doh, “The service modeling process based on use case refactoring,” in *BIS’07: Proceedings of the 10th international conference on Business information systems*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 108–120.

[128] Web ontology language semantic markup for web services (owl-s). [Online]. Available: http://www.w3.org/Submission/OWL-S/

[129] F. Casati, S. Ilnicki, L.-J. Jin, and M.-C. Shan, “An open, flexible, and configurable system for service composition,” 2000, pp. 125 –132.

[130] H. Sun, X. Wang, B. Zhou, and P. Zou, “Research and implementation of dynamic web services composition,” in *Advanced Parallel Processing Technologies*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003, vol. 2834, pp. 457–466, 10.1007/978-3-540-39425-9_54. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-39425-9_54

[131] I. Foster and C. Kesselman, “Computational grids,” in *Vector and Parallel Processing - VECPAR 2000*, ser. Lecture Notes in Computer Science, J. Palma, J. Dongarra, and V. Hernández, Eds. Springer Berlin / Heidelberg, 2001, vol. 1981, pp. 3–37, 10.1007/3-540-44942-6_2. [Online]. Available: http://dx.doi.org/10.1007/3-540-44942-6_2

[132] T. Erl, *SOA Design Patterns*, 1st ed. Prentice Hall PTR, Jan. 2009.

[133] X. Ye, “Towards a reliable distributed web service execution engine,” in *Web Services, 2006. ICWS ’06. International Conference on*, 18-22 2006, pp. 595 –602.

[134] A. L. Lopes, “Executing semantic web services with a context-aware service execution agent,” in *AAMAS’07/SOCASE’07: Proceedings of the 2007 AAMAS international workshop and SOCASE 2007 conference on Service-oriented computing*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 1–15, translator-Botelho, Luís Miguel.

[135] S. McIlraith, T. Son, and H. Zeng, “Semantic web services,” *Intelligent Systems, IEEE*, vol. 16, no. 2, pp. 46 – 53, mar-apr 2001.

[136] M. B. Juric, A. Sasa, B. Brumen, and I. Rozman, “Wsdl and uddi extensions for version support in web services,” *J. Syst. Softw.*, vol. 82, no. 8, pp. 1326 – 1343, 2009.

[137] T. Berners-Lee, J. Hendler, and O. Lassila, “The semantic web,” *Scientific American*, May 2001.

[138] “Resource description framework (rdf),” W3C RDF Working Group, Feb 2004. [Online]. Available: http://www.w3.org/RDF/

[139] D. Brickley and R. Guha, “Rdf vocabulary description language 1.0: Rdf schema,” Online, February 2004. [Online]. Available: http://www.w3.org/TR/rdf-schema/

[140] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. T. Schmidt, A. Sheth, and K. Verma, “Web service semantics wsdls,” Online, November 2005. [Online]. Available: http://www.w3.org/Submission/WSDL-S/

[141] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. K. D. Martin, S. McIlraith, D. McGuinness, J. Su, and S. Tabet, “Semantic web services framework (swsf) overview,” September 2005. [Online]. Available: http://www.w3.org/Submission/SWSF/

[142] J. de Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, U. Keller, M. Kifer, B. Konig-Ries, J. K. R. Lara, H. Lausen, E. Oren, A. Polleres, D. Roman, J. Scicluna, and M. Stollberg, “Web service modeling ontology (wsmo),” Online, June 2005. [Online]. Available: http://www.w3.org/Submission/WSMO/

[143] C. Jacob, H. Pfeffer, S. Steglich, L. Yan, and M. Qifeng, “A view-based approach for semantic service descriptions,” *Next Generation Mobile Applications, Services and Technologies, International Conference on*, vol. 0, pp. 213–221, 2008.

[144] M. Uschold, M. Gruninger, M. Uschold, and M. Gruninger, "Ontologies: Principles, methods and applications," *KNOWLEDGE ENGINEERING REVIEW*, vol. 11, pp. 93—136, 1996. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.111.5903

[145] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing web services on the semantic web," *The VLDB Journal*, vol. 12, no. 4, pp. 333–351, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?id=953243

[146] F. Ramparany and L. Vercouter, "Flexible composition of smart device services," in *In: The 2005 International Conference on Pervasive Systems and Computing(PSC-05), Las Vegas*, 2005, pp. 27 – 30.

[147] J. Rao and X. Su, "A survey of automated web service composition methods," *In Proceedings of the first Internationsl Workshoo on Semantic Web Services and Web Process Composition,SWSWPC2004*, pp. 43–54, 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.90.9880

[148] M. Sheshagiri, N. M. Sadeh, and F. G, "Using semantic web services for context-aware mobile," in *MobiSys 2004 Workshop on Context Awareness Applications*, 2004.

[149] S. Dustdar and W. Schreiner, "A survey on web services composition," *Int. J. Web Grid Serv.*, vol. 1, no. 1, pp. 1–30, 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1358538

[150] I. Robinson, E. Newcomer, M. Feingold, and R. Jeyaraman, *OASIS Web Services Coordination Version 1.2*, OASIS Std., Rev. 1.2, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec.html

[151] D. Bunting, M. Chapman, O. Hurley, M. Little, J. Mischkinsky, E. Newcomer, J. Webber, and K. Swenson, "Web services composite application framework (ws-caf) ver1. 0," 2003.

[152] I. BEA, “Microsoft: Web services transactions (ws-transactions),” 2002.

[153] M. Keidl and A. Kemper, “Towards context-aware adaptable web services,” in *WWW Alt. ’04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. New York, NY, USA: ACM, 2004, pp. 55–65.

[154] T. Strang and C. Linnhoff-Popien, “A context modeling survey,” in *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[155] M. Little, E. Newcomer, and G. Pavlik, “Web services context specification (ws-context),” *OASIS Committee Draft v. 0.8*, 2004.

[156] S. Ran, “A model for web services discovery with qos,” *SIGecom Exch.*, vol. 4, no. 1, pp. 1–10, 2003.

[157] X. Li, Y. Fan, J. Wang, L. Wang, and F. Jiang, “A pattern-based approach to development of service mediators for protocol mediation,” in *WICSA ’08: Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 137–146.

[158] N. Milanovic and M. Malek, “Current solutions for web service composition,” *IEEE Internet Computing*, vol. 8, no. 6, pp. 51–59, 2004.

[159] W. van der Aalst, A. H. M. T. Hofstede, and M. Weske, “Business process management: A survey,” in *Proceedings of the 1st International Conference on Business Process Management, volume 2678 of LNCS*. Springer-Verlag, 2003, pp. 1–12.

[160] F.Leymann, “Web services flow language (wsfl),” Technical Report, IBM, May 2001.

[161] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon, December 2004. [Online]. Available: http://www.w3.org/TR/ws-cdl-10/

[162] A. Bucchiarone and S. Gnesi, "A survey on services composition languages and models," in *International Workshop on Web Services Modeling and Testing (WS-MaTe 2006)*, 2006.

[163] B. Benatallah, M. Dumas, Q. Z. Sheng, and A. H. H. Ngu, "Declarative composition and peer-to-peer provisioning of dynamic web services," in *In Proc. 18 th International Conference on Data Engineering*, 2002.

[164] B. Srivastava, "Planning with workflows - an emerging paradigm for web service composition," in *In Proceedings of the ICAPS-2004 Workshop on Planning and Scheduling for Web and Grid Services*, 2004, pp. 78 – 85.

[165] A. Urbieta, G. Barrutieta, J. Parra, and A. Uribarren, "A survey of dynamic service composition approaches for ambient systems," in *Proceedings of the 2008 Ambi-Sys workshop on Software Organisation and Monitoring of Ambient Systems*. Quebec City, Canada: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–8. [Online]. Available: http://portal.acm.org/citation.cfm?id=1413927.1413928

[166] V. Stricker, K. Lauenroth, P. Corte, F. Gittler, S. de Panfilis, and K. Pohl, *Towards the Future Internet - Emerging Trends from European Research*. IOS Press, 2010, vol. 0, no. ISBN 978-1-60750-538-9, ch. Creating a Reference Architecture for Service Based Systems - A Pattern Based Approach, pp. 149 – 160.

[167] B. Orriëns, J. Yang, and M. P. Papazoglou, "A framework for business rule driven service composition," in *In Proceedings of the Fourth International Workshop on Conceptual*. Springer, 2003, pp. 14–27.

[168] S. Schmid, L. Eggert, M. Brunner, and J. Quittek, "Turfnet: An architecture for dynamically composable networks," in *Autonomic Communication*, ser. Lecture

Notes in Computer Science, M. Smirnov, Ed. Springer Berlin / Heidelberg, 2005, vol. 3457, pp. 94–114.

[169] M. Beer, M. d'Inverno, N. R. Jennings, M. Luck, C. Preist, and M. Schroeder, "Negotiation in multi-agent systems," http://eprints.ecs.soton.ac.uk/3858/, 1999. [Online]. Available: http://eprints.ecs.soton.ac.uk/3858/

[170] M. Wooldridge and S. Parsons, "On the use of logic in negotiation," in *In Proceedings of the Workshop on Agent Communication Languages*, 2000.

[171] I. Brandic, S. Venugopal, M. Mattess, and R. Buyya, "Towards a meta-negotiation architecture for sla-aware grid services," 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.144.5602

[172] R. Kumar, A. Haber, A. Yazidi, and F. Reichert, "Towards a relation oriented service architecture," in *COMSNETS'10: Proceedings of the 2nd international conference on COMmunication systems and NETworks*. Piscataway, NJ, USA: IEEE Press, 2010, pp. 452–459.

[173] P. F. Pires, M. R. F. Benevides, and M. Mattoso, "Building reliable web services compositions," in *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*. London, UK: Springer-Verlag, 2003, pp. 59–72.

[174] R. Kumar, "Fifth generation networking principles for a service driven future internet architecture," *Wireless Personal Communications*, vol. 55, 2010. [Online]. Available: http://www.springerlink.com/content/u330180x5720kn96/

[175] P. Bellavista and A. Corradi, *The Handbook of Mobile Middleware*. Auerbach Publications, 2006.

[176] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping performance and dependability attributes of web services," in *Web Services, IEEE International Conference on*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 205–212.

[177] Y. Wang and J. Vassileva, "Toward trust and reputation based web service selection: A survey," 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.6365

[178] Z. Malik and A. Bouguettaya, "Reputation bootstrapping for trust establishment among web services," *IEEE Internet Computing*, vol. 13, no. 1, pp. 40 – 47, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1495880

[179] D. W. Walker, L. Huang, O. F. Rana, and Y. Huang, "Dynamic service selection in workflows using performance data," *Sci. Program.*, vol. 15, no. 4, pp. 235–247, 2007.

[180] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour, "Semantic ws-agreement partner selection," in *WWW '06: Proceedings of the 15th international conference on World Wide Web*. New York, NY, USA: ACM, 2006, pp. 697–706.

[181] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl - the planning domain definition language," Tech. Rep., 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212

[182] E. Silva, L. Pires, and M. van Sinderen, "A framework for the evaluation of semantics-based service composition approaches," in *Web Services, 2009. ECOWS '09. Seventh IEEE European Conference on*, 2009, pp. 66–74. [Online]. Available: 10.1109/ECOWS.2009.23

[183] F. Belqasmi, R. Glitho, and R. Dssouli, "Ambient network composition," *Network, IEEE*, vol. 22, no. 4, pp. 6–12, 2008. [Online]. Available: 10.1109/MNET.2008.4579765

[184] C. Kappler, P. Poyhonen, M. Johnsson, and S. Schmid, "Dynamic network composition for beyond 3g networks: a 3gpp viewpoint," *Network, IEEE*, vol. 21, no. 1, pp. 47–52, 2007. [Online]. Available: 10.1109/MNET.2007.314538

[185] “The state of the internet operating system - o’reilly radar,” March 2010. [Online]. Available: http://radar.oreilly.com/2010/03/state-of-internet-operating-system.html

[186] R. Schaller, “Moore’s law: past, present and future,” *Spectrum, IEEE*, vol. 34, no. 6, pp. 52 – 59, jun. 1997.

[187] I. Maathuis and W. Smit, “The battle between standards: Tcp/ip vs osi victory through path dependency or by quality?” in *Proceedings of the 3rd IEEE Conference on Standardization and Innovation in Information Technology*, T. Egyedi, K. Krechmer, and K. Jakobs, Eds. Piscataway, NJ: IEEE, 2003, pp. 161–176. [Online]. Available: http://doc.utwente.nl/46343/

[188] N. S. Nappinai, “Cyber crime law in india: Has law kept pace with emerging trends? an empirical study,” *Journal of International Commercial Law and Technology*, vol. 5, no. 1, pp. 22 – 28, Dec. 2009. [Online]. Available: http://www.jiclt.com/index.php/jiclt/article/view/97

[189] K. Yadav, V. Naik, A. Singh, P. Singh, P. Kumaraguru, and U. Chandra, “Challenges and novelties while using mobile phones as ict devices for indian masses: short paper,” in *Proceedings of the 4th ACM Workshop on Networked Systems for Developing Regions*. San Francisco, California: ACM, 2010, pp. 1–2. [Online]. Available: http://portal.acm.org/citation.cfm?id=1836011

[190] P. C. Ensign, N. P. Robinson, and L. Fournier, “BlackBerry in red china: Research in motion navigates institutional barriers in an emerging marketCase comment: RIM in china,” *Thunderbird International Business Review*, vol. 50, no. 2, pp. 129–142, 2008. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/tie.20184/abstract

[191] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Doctoral dissertation, University of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[192] T. D. Huynh, N. R. Jennings, and N. R. Shadbolt, "An integrated trust and reputation model for open multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 2, pp. 119–154, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1146543.1146545

[193] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decis. Support Syst.*, vol. 43, no. 2, pp. 618–644, 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1225716

[194] W. Vambenepe, C. Thompson, V. Talwar, S. Rafaeli, B. Murray, D. Milojicic, S. Iyer, K. Farkas, and M. Arlitt, "Dealing with scale and adaptation of global web services management," in *ICWS '05: Proceedings of the IEEE International Conference on Web Services.* Washington, DC, USA: IEEE Computer Society, 2005, pp. 339–346.

[195] R. Braden, D. Clark, S. Shenker, , and J. Wroclawski, "Developing a next-generation internet architecture," Whitepaper, July 2000.

[196] M. Johnsson, B. Ohlman, A. Surtees, R. Hancock, P. Schoo, K. Ahmed, F. Pittmann, R. Rembarz, and M. Brunner, "A future-proof network architecture," in *Mobile and Wireless Communications Summit, 2007. 16th IST*, 2007, pp. 1–5. [Online]. Available: 10.1109/ISTMWC.2007.4299192

[197] C. Simon, R. Rembarz, P. Paakkonen, H. Perkuhn, C. Bento, N. Akhtar, R. Aguero, T. Katona, and P. Kersch, "Ambient networks integrated prototype design and implementation," in *Mobile and Wireless Communications Summit, 2007. 16th IST*, 2007, pp. 1–5. [Online]. Available: 10.1109/ISTMWC.2007.4299300

[198] C. S. Carr, S. D. Crocker, and V. G. Cerf, "Host-host communication protocol in the arpa network," in *AFIPS '70 (Spring): Proceedings of the May 5-7, 1970, spring joint computer conference.* New York, NY, USA: ACM, 1970, pp. 589–597.

[199] R. Atkinson, S. Bhatti, and S. Hailes, “Ilnp: mobility, multi-homing, localised addressing and security through naming,” *Telecommunication Systems*, vol. 42, no. 3, pp. 273–291, Dec. 2009. [Online]. Available: http://dx.doi.org/10.1007/s11235-009-9186-5

[200] D. Trossen, B. Briscoe, P. M. K. Sollins, L. Zhang, P. Mendes, S. Hailes, B. Jerman-Blaciz, and D. Papadimitrou, “Eiffel report: Starting the discussion,” EIFFEL think tank, Tech. Rep., 2009.