“Open-Source, Web-Based, Framework for Integrating Applications with Social Media Services and Personal Cloudlets”

Deliverable D2.2
Mobile Cloud Analysis

Workpackage: WP2 – Use Cases Analysis and Requirements Specification

Authors: Leigh Griffin (WIT), Dónal McCarthy (WIT), Eric Robson (WIT), Robert Kleinfield (FOKUS), Lukasz Radziwonowicz (FOKUS)

Status: Final

Date: 31/01/2013

Version: 1.0

Classification: Public

Disclaimer:
The OPENi project is co-funded by the European Commission under the 7th Framework Programme. This document reflects only authors’ views. EC is not liable for any use that may be done of the information contained therein.
OPENi Project Profile

Contract No.: FP7-ICT-317883

Acronym: OPENi
Title: Open-Source, Web-Based, Framework for Integrating Applications with Social Media Services and Personal Cloudlets
URL: www.openi-ict.eu
Start Date: 01/10/2012
Duration: 30 months

Partners

Waterford Institute of Technology
Coordinator
Ireland

National Technical University of Athens (NTUA), Decision Support Systems Laboratory, DSSLab
Greece

Fraunhofer-Gesellschaft Zur Foerderung Der Angewandten Forschung E.V
Germany

INFORMATICA GESFOR SA
Spain

AMBIESENSE LTD
UK

VELTI SA
Greece

BETAPOND LIMITED
Ireland
# Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author (Partner)</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td>29/11/2012</td>
<td>Leigh Griffin</td>
<td>Initial table of contents</td>
</tr>
<tr>
<td>0.20</td>
<td>10/12/2012</td>
<td>Leigh Griffin, Dónal McCarthy, Eric Robson, Lukasz Radziwonowicz, Robert Kleinfield</td>
<td>First draft for input at Athens meeting including initial consumer and provider viewpoint and state of the art in software engineering. Initial spec of the cloudlet</td>
</tr>
<tr>
<td>0.30</td>
<td>17/01/2013</td>
<td>Leigh Griffin, Dónal McCarthy, Eric Robson, Lukasz Radziwonowicz, Robert Kleinfield</td>
<td>Second draft fleshing out the features of the cloudlet from partner comments. Provider and Consumer sections completed.</td>
</tr>
<tr>
<td>0.40</td>
<td>20/01/2013</td>
<td>Leigh Griffin, Dónal McCarthy, Eric Robson, Lukasz Radziwonowicz, Robert Kleinfield</td>
<td>First final version released for formal review by Betapond and other interested partners.</td>
</tr>
<tr>
<td>0.50</td>
<td>29/01/2013</td>
<td>Leigh Griffin, Dónal McCarthy, Eric Robson, Lukasz Radziwonowicz, Robert Kleinfield</td>
<td>Reviewer comments addressed, additional section on cloud platforms included. Conversion into template format.</td>
</tr>
<tr>
<td>1.0</td>
<td>31/01/2013</td>
<td>Leigh Griffin, Eric Robson, Fenareti Lampathaki</td>
<td>Final review comments provided by NTUA and WIT and addressed.</td>
</tr>
</tbody>
</table>
Executive Summary

This deliverable forms part of an overall analysis on the state of cloud computing, specifically looking at the Mobile Cloud. The deliverable examines the mobile cloud from the perspective of the end user and the provider. Attitudes towards computing have changed dramatically in the last ten years with technology becoming affordable and more mobile, bringing about a generation of technology savvy users. The availability of technology is complemented by advances in the underlying network, with consistent connection speeds and coverage reaching saturation levels. This has ensured a smooth experience for users and consequently expectations about what technology can do for a user’s life has risen. This expectation has been facilitated by a multi-billion dollar industry delivering applications and services for user consumption. This industry has culminated in the rise of modern social networks, instantly connecting friends and family regardless of geographic location and allowing a hereto unseen level of interaction. This interaction can cause issues from the perspective of the provider, with infrastructure required to cope with the demand.

The model for providers infrastructure however has changed with the emergence of cloud computing. The previous model of purchasing expensive servers was superseded by hiring virtual devices and platforms, with charging mechanisms becoming more open in nature rather than a capital investment. The charging mechanisms vary, with usage often a metric. This change in charging has led to a more careful analysis of the tools and technologies that providers and developers use, with a greater emphasis on scalability and efficiency at the programming level. Additionally, this generation of users is more aware of their privacy footprint and their data and how the world interacts around them. Providers have had to take this into account with technology previously deployed in domains outside of their realm being evaluated and incorporated on their platforms stack. The environment of the mobile based cloud is changing as rapidly as user requirements. This report analyses the current state of the cloud and makes a prediction as to the future of cloud computing and where the OPENi platform may come into it. The culmination of this analysis is the concept of the OPENi Cloudlet, a mechanism for interacting with cloud based services and securely managing a user’s private information and cloud computing experience.
# Table of Contents

1 Introduction ............................................................................................................. 7  
   1.1 Purpose and Objectives ...................................................................................... 7  
   1.2 Methodology ....................................................................................................... 7  
   1.3 Structure of the Document .................................................................................. 8  
2 Term Definition ....................................................................................................... 9  
   2.1 Mobile Cloud Computing .................................................................................. 9  
   2.2 Internet Anywhere / Everywhere ...................................................................... 9  
   2.3 Ubiquitous Computing ...................................................................................... 9  
   2.4 User Centric ...................................................................................................... 9  
   2.5 Mobile Device ................................................................................................... 9  
   2.6 Service Enabler ................................................................................................. 9  
   2.7 Cloud Platform .................................................................................................. 9  
   2.8 Mobile Computing ............................................................................................ 10  
   2.9 Mobile Security ................................................................................................ 10  
   2.10 Cloud Computing ........................................................................................... 10  
   2.11 Cloud Computing Security ............................................................................. 10  
   2.12 Cloud Storage ................................................................................................. 11  
   2.13 Service Provider ............................................................................................. 11  
   2.14 Consumer ........................................................................................................ 11  
   2.15 Service ............................................................................................................ 11  
   2.16 Service Style ................................................................................................... 11  
3 Stakeholders ............................................................................................................ 12  
   3.1 Consumers ......................................................................................................... 12  
      3.1.1 Mobile .......................................................................................................... 12  
      3.1.2 Cloud ........................................................................................................... 13  
   3.2 Providers ............................................................................................................ 15  
      3.2.1 Models ......................................................................................................... 15  
      3.2.2 Frameworks ................................................................................................. 16  
4 Web Based Social Networking: Different Styles ..................................................... 18  
   4.1 Semantically Enhanced Groups ........................................................................ 19  
5 Data Management in the Cloud ............................................................................ 22  
6 The Future of Mobile Cloud Computing: The Cloudlet .......................................... 25  
   6.1 Consumer .......................................................................................................... 25
6.2 Developer ................................................................................................................. 25

7 Future Trends ............................................................................................................. 27
  7.1 Evolution .................................................................................................................. 27
  7.2 Revolution ............................................................................................................... 27
  7.3 Prediction .................................................................................................................. 28

8 A Methodology for Architecting a Cloudlet ......................................................... 30
  8.1 Design Patterns ....................................................................................................... 30
  8.2 Language Paradigms ............................................................................................... 31
  8.3 Concurrency ............................................................................................................. 32
  8.4 Domain Specific Languages ..................................................................................... 33
  8.5 Social Computing User Data Management .......................................................... 33
  8.6 Policy Based Network Management ......................................................................... 36
     8.6.1 XACML .............................................................................................................. 36
     8.6.2 Policy Authoring and Conflict ......................................................................... 37
     8.6.3 Policy Performance ............................................................................................ 37
     8.6.4 Policy Representation ....................................................................................... 38
     8.6.5 PBNM Summary ............................................................................................... 39

9 Conclusion and Recommendations .............................................................................. 40

Annex I: References ........................................................................................................ 41
1 Introduction

1.1 Purpose and Objectives

Usage patterns for Internet services and applications have evolved dramatically in the first decade of this century. In the earlier years, usage was dominated by person-to-service interaction, most often via browsing conventional static web sites through desktop computers. Rapid advances in network and computer infrastructure, and particularly the evolution of mobile, smart, always-connected devices, have produced a dramatic shift in usage patterns. This has culminated in the phenomenal rise of social networks and service computing with such services offered through cloud computing. These represent a significant change in the nature of how users view and access services and applications. This report will investigate the rise of mobile based services from the perspective of a user and a provider. A best practice with respect to development tools for authoring services and cloud based infrastructure is also included. The objective of this report is to discover trends and pointers of a potential paradigm shift to mobile cloud computing and to inform the development of the underlying OPENi architecture.

1.2 Methodology

The methodology adopted for this report took the form of three steps:

Step 1: Analysing the attitudes of users to cloud based services and the rise of the application driven social networking environment which user have grown accustomed to.

Step 2: Analysis of the changes cloud computing has brought to application developers and providers. This is taken from the point of view of infrastructure and tools used, charting the evolution of best practices and approaches to developing cloud based infrastructure.

Step 3: An attempt to predict the future trends of cloud computing based on the current evolution of best practices. This analysis helps to form the future of cloud computing for both providers and end users, the OPENi Cloudlet.
1.3 Structure of the Document

The structure of the document is as follows:

- Section 1 serves as the Introduction to the deliverable outlining the purpose of the work and the methodology employed.
- Section 2 provides some background information on the Stakeholders that the OPENi project will target, namely the Consumers and the Providers.
- Section 3 presents an overview of Web Based Social Networking, outlining the different styles of applications available within the cloud and the challenges that they can bring about.
- Section 4 looks at how user data is managed in the cloud.
- Section 5 provides a comprehensive state of the art on current best practices for design and implementation strategies that platform developers and providers are adopting to meet the emerging demands of mobile cloud computing.
- Section 6 charts out the future trends of Cloud Computing making carefully analysed predictions on the future of the paradigm.
- Section 7 outlines the OPENi vision for the future of mobile cloud computing, the Cloudlet.
- Section 8 concludes this deliverable.
2 Term Definition

2.1 Mobile Cloud Computing
Mobile cloud computing is the converging of smart phones and cloud computing. Users currently use mobile devices with limited computing power, bandwidth and other resources. Cloud based software can overcome these device restrictions by offering an end user a platform on which to run computationally intensive applications and a means to store their data. It offers opportunities for mobile operating systems, the way we perceive processes and data and device communication. Mobile cloud computing enables devices to access any computational intensive services at any-time and anywhere on any device, as long as the bandwidth requirements are met. Localized clouds can be used in order to reduce latency. Cloud computing benefits from the additional context and personal information typically provided in the mobile context.

2.2 Internet Anywhere / Everywhere
Devices that are (almost) always connected to the internet either via mobile internet opportunities or LAN networks. New use-cases and different opportunities are emerging.

2.3 Ubiquitous Computing
Computing everywhere.

2.4 User Centric
An approach based on (end user) use-cases, personas, their needs, goals, knowledge and experience. This includes analysis, design, development, testing and evaluation.

2.5 Mobile Device
A device that is portable and capable of accessing services through a network.

2.6 Service Enabler
Building block to enable the development of higher order services while not being an end user service itself.

2.7 Cloud Platform
The term cloud platform is used to refer to any computational or storage cloud. Cloud is used as a Synonym.
2.8 Mobile Computing

Mobile Computing is any computation on a mobile device. The term includes hardware, software and network aspects of mobile devices. The mobile device hardware includes sensors and offers new iteration moralities such as NFC or voice commands. The software is enriched by the additional hardware input and benefits from the social context and trust with personal data from the user. Mobile networks achieve higher bandwidth and enable the development of more network intensive remote services.

2.9 Mobile Security

Mobile devices, specifically smart phones have become a major part of our daily life. They collect and use a variety of personalized data with or without the users knowledge. This information includes monetary, social, professional or localization data. The device also represents a single point of failure and is connected to the internet. As many mobile devices are application platforms, the trade-off between security concerns and access to personal data or functionality is often left to the user. The permission frameworks are coarse grain and do not offer enough inside into the applications intend.

2.10 Cloud Computing

Cloud computing refers to the concept of virtualized computational resources which can be used on demand. As with cloud storage, the infrastructure is abstracted and not known to the customer. Cloud computing is often tied to cloud storage in order to persist data. Several different forms of cloud computing such as Infrastructure (IaaS), Platform (PaaS) or Software (SaaS) as a Service exists. The difference between them is the abstraction layer which is available to the customer through the service API. While IaaS exposes a virtual machine based service API that enables customers to allocate machines with different specs, PaaS provides a platform API to the developer for which Applications can be developed. SaaS provides instances of software, chosen by the provider. The user rents the software and is able to configure it.

2.11 Cloud Computing Security

Cloud computing security encompasses a variety of concerns. From a technical perspective for example, virtualized hardware (IaaS) introduces a new layer, the hyper-visor, which offers additional attack vectors. The security of this layer is not influenced or watched by the consumer. Whether a company takes the appropriate security measures is based on trust. Servers based on virtual machines can be attacked from a hyper-visor layer without disrupting the machine and without the knowledge of the user. Any server-side encryption or security can be circumvented by peeking at the machines memory. Due to law, some companies may even have to offer silent access to these running virtual machines. Physical servers on the other hand can be locked up and checked via a video stream. Similar concerns can be raised for PaaS and SaaS.
2.12 Cloud Storage
Cloud storage refers to the concept of virtualized storage resources which can be used on demand. The abstract storage is offered as a service via the network. The hardware resources are typically not known to the customer and may be heterogeneous. They may span across multiple servers but are represented as one coherent form of storage through a service API. The virtualized storage is provided on-demand and can be configured by the user.

2.13 Service Provider
Service providers include online service providers and traditional transaction based services, offer applications, tools and facilities to consumers with the goal of maximising their number of customers while reducing operational expenditure.

2.14 Consumer
The end user who consumes a service at a cost. The cost is dictated by the provider and may be tangible (money based) or intangible advertising / marketing driven.

2.15 Service
A service can be a standalone application or a tool to assist another service or application.

2.16 Service Style
A service style is a broad classification for services. Each service style will have certain characteristic including traffic profile, resource footprint and responsiveness expectations from the consumer side.
3 Stakeholders

A number of stakeholders are involved in the OPENi project. This section talks about two major stakeholders, the Consumer and the Provider.

3.1 Consumers

3.1.1 Mobile

In the last ten years the service landscape of the consumer web has grown in size and diversity. Today's cloud based services provide many of the functions which were once limited to personal computers. Working online on office documents or watching videos are examples. The online nature of data and services allows users to consume them anytime, anywhere and share them with others. Many cloud based services (e.g. Flicker, YouTube, LinkedIn, Dropbox, etc.) provide a public API to include them into other services or applications. Such an API can be used to address new use cases or create a better user experience. Web-based interfaces make them independent from the user's hard and software and accessible by any web-enabled device.

Personal data and the ability to connect and communicate with other people are central to many cloud-based services. This includes social networks, which are not only cloud based services but platforms as well. These platforms encompass the user's personal and even professional life. They represent their digital identity. Everything between humans, from the communication to complex interactions such as games can be provided through a social network. Since social networks hold the user's personal data, they are able to provide customized services. Their widespread use and the ability to encompass social data make them an appealing service platform. In return, the platform is constantly extended through third party services. This provides more features to the user and enriches his experience. In addition, social networks extend their features as well. For example, Facebook added a chat, (Business) Fan Pages and the Like Button. This makes them the perfect hub to conveniently share information and consume personalized services. However, social service platforms are often seen as closed systems by a user, this is not the case however. A detailed description and insight into the data that is connected to an account can be retrieved by a user if they look in the right places. Similarly information about how his data is used with respect to applications is readily available. Users however are more interested in how the application works and what they can access to worry about their data storage or what information they are releasing [1]. The revenue of platforms like Google and Facebook is generated in connection with user data. Tracking, personalized advertising or the access of partnering companies are some example. This conflicts with the users wish and right for privacy. Facebook for example has recently reduced the privacy guaranties they provide through their terms of service (ToS). Face recognition is just one example where Facebook and the EU data protection agencies have been in conflict. Closed platforms also hinder interoperability and lead to a certain degree of vendor lock-in. Users are not able to move between these platforms or connect to the users of another.
The market-share of feature-rich, web-enabled, connected mobile devices, such as smartphones, has rapidly increased in the last five years. The user experience when consuming cloud based services has changed due to the devices mobility. Services are accessible anywhere at any time. Mobile devices provide context data such as location data and personal data like the calendar to further enrich services. They are application platforms which often serve as frontend to cloud based services. Using a map application does not require the user to locate his current position. Applications and their corresponding cloud based service can locate the user via GPS and display his current position. Once a service is allowed to use the GPS, it can do so without the user's interaction. Users benefit from the personal and location aware nature of smartphones. The design of applications encompasses the online functionality to support the daily needs and routines of users. They also benefit from the omnipresence of the devices as he is always connected.

App-Stores to conveniently download applications are integrated into all major smartphone systems (Apples App-, Google Play! and Microsoft's Windows Store). Mobile devices, like social networks, are personal hubs. System vendors like Apple, Microsoft and Google all integrate their cloud based services into the operating system, through applications or local services. Smartphones and their usage are bound to an identity, as soon as cloud based vendor services like the App-Store are accessed. This provides personalized services and streamlined interactions. Once a credit card is registered the user can buy at the App-Store with a few button clicks. Vendor platforms have become a service platform as well. They, among other services, support billing, data synchronization and media services. Platforms like Google also support access the user's data by third party services.

Users have become familiar with the always connected nature of smartphones. They support their daily needs and routines. Social media and vendor platforms are service platforms. They hold the personal data which enables personalized and user centric services. Users benefit from these platforms, as they provide the needed data to the cloud based services. Mobile devices are the connection between the physical user and his digital identity. They provide a variety of sensors and input schemas which can be used by applications, and therefore by cloud based services. These four perspectives, the hardware, the local applications, the service platform and the cloud based service landscape provide the mobile consumer experience.

### 3.1.2 Cloud

Outsourcing is a common strategy of cost reduction. As software has the ability to run on any compatible computer, computers can be outsourced to any place in the world. Achieving low prices or low latency are common reasons for outsourcing hardware. However, in order to guarantee the availability during peak loads, companies that provide large scale services have to overprovision their hardware. Cloud computing developed in response to this waste of resources. Clouds are multi-tenancy platforms that provide virtualized resources on demand. Clients self-provision their resources and pay for what they use, the so called pay-as-you-go model. Cloud platforms are services. They provide API, tools and web-platforms to manage the service. A cloud customer can request additional resources such as machines or storage on demand. The machine can be used as soon as it is set up.
Once the user does no longer need the machine, the billing stops and the instance is destroyed. The cost is determined by the resources that were used. This among others may include the bandwidth, the processor and the amount of memory or storage. Persistent storage is provided by storage services. Their cost model involves the number of data accesses, the used bandwidth and the capacity. An example for such storage is Amazons S3. Instead of providing virtual machines, other cloud providers rent out scalable software or let users develop and run applications on their distributed framework. Clouds are especially attractive for businesses with a varying load, which benefit from the simplified or automated balancing capabilities. They also reduce the cost of maintenance, as the hardware is maintained by the provider. In case of a system failure, virtual resources are automatically migrated to another host server. Downtimes are reduced since there is no need to wait for the hardware failure to get resolved.

Cloud services are a potential cost reduction and a simple way to temporarily scale services depending on the customer’s needs. However, the loss of control and the availability, durability, consistency of the service and the data are common concerns. Unlike dedicated servers in datacentres, users are not able to secure cloud servers themselves. This means they have to trust the cloud provider with the security of their data. From a legal point this may prove inefficient for certain data, such as medical files. Jurisdictional and privacy concerns within the EU are also common. Legislature, companies and users are aware that while using remote resources, jurisdictions and laws which govern access and security may change. The Patriot Act and the Foreign Intelligence Surveillance Act (FISA), provide US intelligence agencies with a way to circumvent the Safe Harbour agreement. They have access to all cloud data of US-based companies. This includes data centres of such companies even if they are located within the EU. US law has viewer restrictions on the data processing and storage of non-US citizens. On-going legislative EU initiatives are addressing the problem.
3.2 Providers

3.2.1 Models

New applications and services are increasingly being developed so that they can be deployed into the cloud, be that Amazon Web Services (AWS), Microsoft Azure, special purpose clouds such as those emerging for level 3 secure health care data, home grown clouds or clouds provided by other third-party vendors. As services are deployed into third-party clouds the cost of operating these services change in their profile to become more opex (operational expenditure) based, as they typically incur monthly, recurring fees based on their usage of cloud resources. These opex costs drive new requirements for services such as “efficiency” and “quality”, in terms of using cloud resources, as an inefficient service can generate substantial additional, and unneeded, costs. A poor quality service, as determined by resource consumption, may inadvertently spike costs by an order of magnitude whilst handling a relatively small number of clients. As such, one of the key stakeholders in the process, the provider, has changed the model for how they do business. The platform they deploy their services on was the first to change.

Cloud computing is one of the key drivers compelling the current wave of innovation ([2]). For the first time, corporations are moving their sensitive data and operations outside of the building. They are placing mission critical systems into the cloud with computing capacity now metered by usage. Technology challenges are not solved by sinking more capital into powerful computing infrastructure. The model is moving away from the costly Infrastructure as a Service (IaaS), where companies provide physical host machines, to the more economical Platform as a Service (PaaS) ([3]). When Ruby on Rails (RoR) was launched it was a highly innovative development framework ([4],[5]). The key driver for mass adoption of RoR was hugely increased developer productivity through convention over configuration, an approach which has now certainly entered the zeitgeist and which has been adopted by almost all of the current development stacks: for example Python’s Django¹ or PHP’s Cake.² The predominant application deployment model for most organisations during the rise of RoR was owned server infrastructure: i.e. make some capital investment in server hardware on which to deploy applications. Under this model, operational expenditure was relatively static and was based on monthly costs for colocation and bandwidth. Operational efficiency of deployed, in the field, applications was not so important for anyone but the really large sites, as long as the application could scale horizontally to some degree, capacity could be added by purchasing more hardware and making appropriate infrastructural changes, such as clustering or simply buying faster machines ([6]).

With the mass adoption of cloud computing, this model is superseded by PaaS as a more affordable solution. Deploying to the cloud requires little or no capital investment; however, operational expenditure is now directly tied to the efficiency of deployed applications. There is now a clear

¹ https://www.djangoproject.com
² http://cakephp.org/
economic driver for efficient web applications, services and deployment platforms. Addressing the latter allows a platform for the applications and services of the future to run on. Google required fast JavaScript so that its services like Gmail\(^3\) and Google Calendar\(^4\) would work efficiently and render quickly for end-users. To do this, Google developed the V8 JavaScript engine ([7]), which compiles JavaScript into highly optimised machine code on the fly. Google open-sourced the V8 engine and it has been adapted by the open source community for cloud computing. The cloud computing version of V8 is known as Node.js ([8]), a high performance JavaScript environment for servers ([9]). In the PaaS model, an entire computing platform including operating system, execution environments such as Node.js and Vert.X\(^5\) and storage mechanisms are provided in a manner capable of scaling to match the demand. The customisation and modularity of offering such as Heroku\(^6\) and Amazon Web Services\(^7\) (AWS) offer support stacks for applications with complementing language and application choices. With this new costing model, performance becomes important as the operating expense dominates, driving the need for highly efficient, lightweight solutions. Architectural choices can thus dictate whether a domain is viable or non-viable, from a cost point of view.

In Section 6 the technological changes they have embraced will be discussed further.

### 3.2.2 Frameworks

Several cloud computing frameworks or stacks have emerged for deployment by providers. These stacks are usable by the consumers of mobile cloud services and the developers of such services. Understanding the core concepts and design principles behind the frameworks already in deployment and currently undergoing further research can help to better inform the OPENi cloudlet platform. Abstractioning the best practices from a subset of existing frameworks can help to identify trends and requirements that the OPENi framework will have to meet in order for future deployment and subsequent adoption. This section briefly outlines a number of stacks that are available for usage, a more comprehensive description and survey can be found in the work of [10].

- **OpenStack\(^8\)** is a collection of open source components used to deliver public and private clouds. The open nature of the release combined with the frequent updates to components and interoperability with other cloud providers, such as Amazon EC2, has encouraged a considerable uptake by providers and developers.

---

\(^3\) [http://gmail.google.com](http://gmail.google.com)
\(^4\) [https://www.google.com/calendar/](https://www.google.com/calendar/)
\(^5\) [Vert.X](http://vertx.io/)
\(^6\) [http://www.heroku.com/](http://www.heroku.com/)
\(^7\) [http://aws.amazon.com/](http://aws.amazon.com/)
\(^8\) [http://www.openstack.org/](http://www.openstack.org/)
- **Eucalyptus** is a private cloud company that allows cloud infrastructure work around existing legacy systems. It uses API compatibility to allow interaction with external cloud providers such as EC2 and provides a high level management stack for managing the cloud stack and underlying resources.
- **OpenNebula** is a framework for managing distributed data centre infrastructures. The design of this framework is with extensibility in mind, capable of interfacing with existing and new tools through a flexible modular toolkit.
- **Nimbus** is a set of tools dedicated to the scientific community offering a storage cloud compatible with EC2. Simplified management tools control infrastructure services and facilitate the integration of existing clouds.
- **mOSAIC** is a research project developing a means for users to negotiate the use of Cloud based services through an open source API and platform for multiple cloud providers. A brokering mechanism facilitates service matching and composition, breaking down barriers of platform dependent APIs.
- **CloudSpaces** is another research project looking at defining open APIs and standard formats to enable flexible information sharing among personal clouds. Scalable storage and service access are among the core output components of the project to empower an application developer to develop and deploy services to end users personal clouds.

As can be seen from the frameworks outlined there are a number of qualities that any future cloud framework stack must possess. Compatibility is a big issue, being able to hook a framework into existing cloud stacks and providers is a must. A popular means of ensuring compatibility is the usage of API chains to hook into services and provide a mechanism for interaction between provider and user. Flexible tool chains supporting this interaction from all interested parties and an extensible framework that is capable of adapting to future needs and trends of users, developers and providers are also desirable qualities for a cloud framework. The framework produced as part of the OPENi project will have to adhere to these basic qualities. A more comprehensive overview of cloud based services including cloud frameworks and their associated APIs can be found in Task 2.1.
4 Web Based Social Networking: Different Styles

With the users perception of the internet evolving, a new technology savvy generation had emerged into a world that was experiencing a computing paradigm change. High speed broadband networks were emerging, mobile, ubiquitous and pervasive computing brought hereto unseen penetration of connected devices. The original group communication technologies of Instant Messaging, Message Forums and IRC, among others, were being adapted and modified to meet this new generation’s demand. The powerful resources available meant an overhaul of the popular group mechanisms, while never straying far from their original purpose of connecting people and facilitating communication. The modern evolution of group communication technologies is social networking, in particular, web based social networking which takes on a number of forms.

Online Groups are provided by some websites as a place where a group of people can come together and have discussions about common interests and meet like-minded individuals, doing so in a public or private manner. Online Groups were in a sense an evolution of traditional collaboration software, made available publicly and inherently more usable. The term Groupware, another term for collaboration software, is defined by [11], as *intentional group processes plus software to support them*. Proprietary software products such as Lotus Notes\(^{14}\) and Microsoft Exchange\(^{15}\) provided integrated collaboration functionality through a suite of group communication technologies including email, group calendaring and instant messaging. These communication platforms allowed individuals, largely those of a professional nature, to effectively work in a coordinated manner towards a common goal. Service providers such as Google\(^{16}\), Yahoo\(^{17}\) and Windows Live\(^{18}\), offered a free combined group service, mirroring the features of traditional Groupware and making it available to a larger membership. This grouping mechanism was the first such attempts at large scale integrated group communication technologies. The sites offer countless free groups around general topics such as health, sports and news with an abundance of sub groups looking at particular themes.

Web based social networking emerged from online groups with individuals constructing a public or semi-public profile within a bounded system, with the intention of interacting on a more personal level with others. Identifying a list of other users with whom they share a connection, users can view and traverse their list of connections and those made by others within the system ([12]). Their social network is in effect a self-contained group realised through intelligent interfaces and interactions. Timed with a more socially aware generation, several styles of social networking sites, appeared. These include what has come to be known as traditional social networking sites, including Bebo\(^{19}\),

---

\(^{14}\) Lotus Notes http://www-01.ibm.com/software/lotus/products/notes/

\(^{15}\) Microsoft Exchange http://www.microsoft.com/exchange/en

\(^{16}\) https://groups.google.com

\(^{17}\) http://groups.yahoo.com/

\(^{18}\) https://groups.live.com/

\(^{19}\) http://bebo.com
Facebook\textsuperscript{20}, MySpace\textsuperscript{21} and Google+\textsuperscript{22}. Blogging and microblogging social networking sites, such as Twitter\textsuperscript{23}, Blogger\textsuperscript{24}, Wordpress\textsuperscript{25} and Friend Feed\textsuperscript{26}. Service oriented social networking is also another style as characterised initially by Gowalla\textsuperscript{27} and later by FourSquare\textsuperscript{28}. The initial concepts of groups within this generation of social networks was an extension of the standard profile with little to no additional group services offered. The principle idea was to use these groups in order to expand a person’s direct contacts, a mechanism still achieved through their profile, thus using the overall service suite at a profile, rather than group level. The rationale behind keeping interactions at a profile level rather than a group level, is not quite clear. Concerns about scalability is a possible explanation as the volume of users social networking sites attract has caused disruptions in the past, which have been addressed through technology ([13]).

Using Facebook as an example, Facebook’s initial attempts at group integration had three distinct styles, two of which were officially supported and one of which evolved from user behaviour. Networks, where one could join based on profile information such as location, educational institution or employer. These networks served little purpose only to allow an additional means for users to connect with others. The second group style promoted was the like page. Here users could create a page based on a topic or concept and people could join the page by liking it, an act of publicly declaring an interest in a topic such that it appears on the newsfeed of your buddylist. These pages functioned the same as a profile page with the ability to post pictures and messages for everyone to see. Users circumvented the lack of groups by creating a third style, the profile group. In this instance a user would create a new profile, typically promoting an event or business and add friends to their friendlist to create a community via the newsfeed. Facebook responded by introducing events as a separate service and introducing a formal representation for groups, as well as means to recommend groups to others ([14]). Now users can create a formal group and invite others into it. Groups can be public or private and full administrative control is possible. Photos can be shared and private events organised within the group, with richer service consumption not yet provisioned for.

\subsection{4.1 Semantically Enhanced Groups}

Group Communication was originally designed to facilitate participation and discussions among members. Technological advances were made in supporting fields, allowing group communication to leverage some of these features. The user base had also profoundly changed, with the need for users

\begin{itemize}
  \item[20] http://facebook.com
  \item[21] http://myspace.com
  \item[22] http://plus.google.com
  \item[23] http://twitter.com
  \item[24] http://blogger.com
  \item[25] http://wordpress.com
  \item[26] http://friendfeed.com
  \item[27] Now working with Facebook as their check-in service
  \item[28] https://foursquare.com/
\end{itemize}
to stay connected and informed of the status and activities of others in their online address book. This desire for information led to the development of applications with added contextual information, enhancing the user experience. These semantic additions are present within existing group infrastructure and will be more prevalent in emerging social networks as social applications. Social applications refer to a class of applications that integrate with one or more social networks ([15]). Applications on social networks are increasingly becoming a focal point for group formation and a means of generating a group full of like-minded individuals. Masked as community games and retaining the goal of groups, that being a common purpose or task, applications are quickly becoming an additional means of group formation within social networks. Social Applications are designed to be cross platform, accessible on mobile devices, potentially provisioned in the cloud or in standalone hardware and designed to emulate native applications. While the evolution of this trend towards socialisation has added more incentive to participate in groups it has potentially raised more questions about the structure of groups and their long term scalability. The effect of socialisation is that it can cause some applications to become very popular amongst a user base very quickly. This is often referred to as viral spread because the application is passed from one user to others in their social graph ([16]) and then on to friends of friends. The impact of this viral spread on an application is to cause sudden spikes in server load for a web application. Presence, Location and Context are three such services deployed within semantically enhanced groups, which are in turn impacting on how groups are used and how they form.

Presence is a current status indicator showing the ability and willingness of a user to communicate across a range of devices. The presence mechanism is designed to allow a real time update for a person, keeping their friends informed. It is typically employed in instant messaging applications to show a person’s friends list what they are doing. Typical default presence statuses include chat, online, offline, busy, do not disturb, and away. The presence feature can be used for custom messages and many applications integrate with this feature. Messenger programs have developed plugins allowing a user’s music or video choices be echoed as a presence status to the person’s friends list. GPS technology on mobile phones can advertise the persons current location through the presence mechanism. Presence is also at the core of enabling protocols such as XMPP ([17]) and can cause issues when used outside of the intended scope of the protocol ([18]). Privacy concerns with the usage of presence as a service medium ([19]) within the context of an entire friendslist is making groups a more attractive place to direct presence and retain a form of control over the privacy and security of the updates.

The desire of people to stay connected and inform friends and family of their current activity and location has led to the integration of location based technology into devices. Mobile phones, laptops and even watches have the capability to track their location. Services have been built to take this data and use it to inform others, often through a group communication medium such as social networks, posting the information to a newsfeed. Location Based Services, termed Geospatial services ([20] are playing an increasing role in social networks, moving beyond personal usage to become a marketing tool, a gaming mechanism as well as a formal outlet for handling emergencies and announcements. The notion of a check-in is a geolocation announcement to a social group advertising your presence at a particular location. Services have emerged to support users who have
registered their location ([21]), however, the check-in notion has started to evolve away from the original geographic only usage. Checking into events such as sports performances, cinema performances or TV shows\textsuperscript{29} has abstracted the core grouping centric logic from the requirement to physically be present. This has seen a scalability profile not yet associated with mainstream check-in services. It is conceivable, for a popular event, that the average daily check-in metrics of a geographical based service ([22]) could be exceeded within the opening credits of a popular TV show. That initial burst could potentially be handled by the current suite of services and infrastructure in place, but adequate service provisioning to target this newly created group is problematic. For example, layering a service on top of that, such as sending a Quick Response (QR) code with a discount, in order to deliver a marketing campaign, would create an enormous amount of strain on an underlying system.

In recent years the notion of context awareness has emerged in communication research, particularly in the field of ubiquitous computing ([23], [24]). Devices became smarter and started to make assumptions about the user's current situation based on information being polled from the user and the environment. Watches with built in heart monitors, GPS enabled devices (laptop, mobile, watch), accelerometers on mobile phones and wireless health body kits (such as insulin monitors) are examples of context aware devices. These devices are enabling technologies, providing data to services for translation. Services can take this data and abstract understanding from it, providing a feature rich service for the end user. The user's social environment i.e. the co-location of others, their social interaction and group dynamics, as outlined in [25], plays a major role in context awareness. It is often the end consumer of such information as well as a major provider. Intelligence ([26]) abstracted from the physical environment and community groups is driving innovative services from environmental services to urban sensing, such as traffic planning and public safety.

\textsuperscript{29} For example http://getglue.com/ offers a means to check-in to TV, Movies and Music
5 Data Management in the Cloud

Depending in their service abstraction and their focus, cloud platforms consist of different components. They contain a host component to manage the hardware. A virtualization component is used to abstract the underlying hardware and provide a computational service platform. The storage component provides virtualized storage capabilities. They also contain a network component to integrate with virtual switches, routers or control Firewalls or VLANs. A management component lets users configure their resources and provides manage their tenants. A quota component is used to define service limits. A monitoring component provides performance indicators of virtual or hardware resources to users and providers. Cloud platforms contain an accounting component that provides cloud providers with the opportunity to use the platform commercially out of the box and to integrate them in charging and billing platforms. An authentication and authorization component is used to give users the opportunity to self-manage and provision their cloud resources. The AAA-components (Authentication, Authorization and Accounting) enable the pay-as-you-go business model typically found on cloud platforms. To enhance the user experience, a dashboard is provided by many cloud platforms that provides access to information and management functions.

Cloud platforms are multi-tenant services. Multi-tenancy means a single instance of a system is used by multiple clients. It is software specifically written to scale and provide a sandboxed environment to the users. This also means a loss of configurability to the user in order to provide fairness. The platform has to ensure that users are not able to interfere with each other and that resources are used efficiently. Service assurances are provided in the form of services level agreements (SLA).

Cloud platforms provide different forms of abstraction and automation. The amount of automation is directly linked to a platforms level of abstraction. The more a user can configure and impact the system, the less it can assume in order to scale. Three categories of cloud platforms are commonly identified. Infrastructure as a service (IaaS) provides abstract hardware. Platform as a service (PaaS) provides developers with an operating system abstraction, while Software as a service (SaaS) provides configurable software components to the user.

- IaaS abstracts the underlying hardware with virtual machines. The customer uses these virtual machines like any other machine. He is able to install an operating system and run programs on top. Users may load pre-configured images to boot already configured instances or use configuration management software like Chef or Puppet to initialize the machines automatically. The virtual machines run on top of a hypervisors like KVM or XEN. The hypervisor in return runs on the provider’s hardware. They manage the physical resources of a host for the concurrent use by the virtual machines. The term IaaS commonly encompasses the abstraction of storage, network and computational components. The user is able to manage his resources to a varying degree, deepening on the access the platform provides. The self-provisioning means users are able to allocate and release virtual machines, storage or virtual network resources as needed. IaaS platforms can be used to match any use case. However, it does not automatically scale computational resources like virtual machines for the user. The user is responsible to write or use a software component that instantiates virtual machines or do it manually. He is also completely responsible for the interaction of the machines and their software. The user can provide a large
scale cloud based service that is accessed by millions or run an analytical task over a finite number of resources. In case of an analytical task, the user may instantiate a fixed amount of resources and run the task on them. This matches the IaaS platform abstraction perfectly as the allocation of resources does not change and the user is able to pre-allocate the need resources. He can calculate his costs and manage what resources are needed for the job.

- The term **Storage as a Service (STaaS)** has been used to encompass cloud based storage solutions. Storage can be seen as part of the infrastructure. However, storage which is not part of a virtual machine, but rather external (e.g. S3) can be referenced easier. Storage solutions like Google's Cloud SQL and Cloud Storage or Amazon's S3 are examples for large scale multi-tenancy solutions.

- **PaaS** abstracts the underlying hardware with an operating system. The user is able to develop applications for these operating systems also called platforms. Unlike the general purpose operating systems (e.g. those the user may install on an IaaS cloud) these platforms are much more restrictive. They can be compared to browsers and their JavaScript engines. Such systems are commonly referred to as sandboxes. For security reasons, they provide limited access to the underlying resources. The amount of control the user has over the resources is determined by the cloud provider. The user many have access to the allocation of worker threads, or he must respect quotas for his services. A request to the Google App Engine for instance needs to complete within a certain timeframe, or is terminated. Programs for PaaS platforms are written in one or more languages. The platform has to provide a runtime support for these languages. Most runtimes such as Python are not designed as a sandbox or do not scale. The platforms runtime has to be altered to provide for multi-tenancy and scalability. For example, access to the native file system or processing resources may not be granted to the app. Loading wrapped C libraries, common in Ruby and Python, may also not be possible. Google's App Engine is an example of such restrictions. Other providers virtualize the underlying hardware and provide each user with their own virtual environment. Heroku\(^{30}\) for example does not have the same restrictions. Users are able to run native extensions as part of their application. This freedom also comes at a cost. While App Engine code scales by itself, users should indicate dependencies their runtime application will require. This is a good example for the aforementioned dependency between abstraction and automatism. Aside from native libraries, PaaS platforms do not support the installation of applications, such as database servers. The developer is required to use external storage solutions. Often more than one is provided by the same platform.

- **SaaS** abstracts the underlying hardware with a software catalog. The clients may rent any software the provider has in his catalog. Clients who would normally buy a server to provide a mail infrastructure to their workforce can instead rent it as software. SaaS software provides web-based interfaces to manage and use the software. It scales depending on the load, to accommodate the user's demands. The configuration of the software components is streamlined to accommodate most use cases of the software. The streamlining is also required to provide scalable software. As mentioned before, in order to provide automatic scalability the software needs to reduce the possible impact the user may have. A typical mail server for example includes the means to configure the back-end storage or to define mail queues and filters. This

\(^{30}\) [http://www.heroku.com](http://www.heroku.com)
can greatly impact the systems performance. SaaS providers may instantiate software for each user. However, in order to increase scalability and the use of resources, software that supports multi-tenancy is preferred. SaaS can be used by non-technical companies, as the platform maintenance is done by the provider. The initial setup or reoccurring maintenance of the software can be outsourced to an engineer.

The different cloud platform categories do not directly compete with each other. Google and Amazon for instance provide each of them. The categories support different use cases and business models. Each of them has restrictions on configurability and scalability. IaaS platforms can be used to accommodate almost any use case. Users who have specific software requirements are not able to use PaaS or SaaS platforms. They need to instantiate their own virtual machines and install the needed software. SaaS can be used if no specific software, just an implementation to a software type (e.g. a mail server) is needed. PaaS is a solution between SaaS and IaaS. The motivation is to provide an existing and familiar runtime with software components (libraries) to the user. He can use these components to implement custom tasks without setting up a scalable system. IaaS and PaaS are used in connection with storage as service solutions to hold persistent data. While they provide the computational platform, the storage is left to a scalable STaaS.

The user has the choice between different cloud platforms or stacks. Aside from commercially deployed systems provided by Amazon, Google, Microsoft or Salesforce the user is able to deploy his own cloud and become a provider himself. Open source systems are free, extendable, benefit from their community and prevent a vendor lock-in. CloudStack is an Apache incubating project. The initial code has been developed and donated by Citrix. The OpenStack Foundation, a cooperation of companies like Rackspace, Cisco and VMware is developing OpenStack. OpenNebula has been developed as a research platform and has been released an open source project. It is used by CERN. Eucalyptus was initialized by the University of California and is now maintained by Eucalyptus Systems. All of these projects are IaaS platforms. As they are under current development their features are growing. Eucalyptus has been widely deployed due to its early API compatibility with the Amazon Web Service (AWS) platform. Such compatibility is regarded as an important feature for open source cloud platforms. It eases the transition of existing AWS customers and allows them to operate their services on their private infrastructure or to switch to a competitor. OpenStack members however have voiced concerns about the dependency on a vendor specific API. Currently all cloud platforms provide an AWS compatibility in some form. They all support multiple hypervisors and provide the same base functionalities for running and storing virtual machines. They support large scale deployment and offer many of the save management functionalities. Compared to the other projects the OpenStack project is less mature and less widely deployed. However, large companies are part of the OpenStack Foundation and the system is under active development. It however, is represented less as a platform and deployable product and more as a component stack that can be customized.

6 The Future of Mobile Cloud Computing: The Cloudlet

As can be seen from the previous sections, cloud computing is in a constant state of evolution, with providers and consumers providing innovative use cases and applications. The OPENi project envisions the concept of a cloudlet for use by both the consumer and application developer.

6.1 Consumer

A cloudlet is a virtual representation of an individual user’s personal space in the cloud, storing personal data and metadata about the user and applications they use. The customisable nature of the cloudlet allows the end user control what content is to be made available to other cloud based services and providers. The user is primarily a mobile consumer interested in accessing and consuming content served within cloud computing held infrastructure.

6.2 Developer

A cloudlet is a facilitating tool enabling integrated APIs for the management of their users cloud storage and preferences. Developer cloudlets can access shared metadata in their users’ cloudlet space, enabling pattern discovery across their application suite for analytics, quality of service monitoring and the identification of potential new user focused applications.

The cloudlet possesses the following attributes and characteristics:

- A storage mechanism, on a user defined addressable space, for the storage of personal data and metadata of an individual user
- A mechanism for enabling the sharing of information between applications, services and devices by operating as a middle-man service
- Configurable by the user to ensure control over the access granted and the information that is available to other cloudlets and applications
- Interoperable with major service providers through API chains
- Capable of offering a historic perspective on users interactions through the Cloudlet

In OPENi a server side cloudlet platform will exist to manage individual cloudlets owned by users. The cloudlet platform will act as a mechanism to enable an application to access a users' cloudlet and if necessary facilitate the user in the creation of a cloudlet. The platform can be deployed by both providers and large scale developers allowing them to manage a cloudlet infrastructure.

This feature set will be realised through a User Interface and the hosted cloudlets are made available to external applications through discovery mechanisms and APIs. The User Interface will deal primarily with the management of the cloudlet and the configuration of access as required by the user. This access will be facilitated by best practices from the point of view of access control and Task
2.3 provides a more comprehensive overview of access control mechanisms and privacy and security concerns that are relevant to the Cloudlets design. The API layer will govern discoverability, access by applications and storage mechanisms for private data. This feature set combined will offer a lightweight, scalable and customisable access management to a user’s cloudlet.
7 Future Trends

7.1 Evolution

The evolution of the Web has been well charted in recent years. With the coining of the term Web 2.0 (2004) a useful starting point, there is a reasonably comprehensive understanding of how the web as a platform has progressed since. This has included the stabilisation of web services protocols, the rise of User Generated Content, the proliferation of Mobile web, the advent of the Smart Phone/Device/Tablet and the App Store Model.

However, the underlying tools, architectures and development practices have also been through an overhaul over this period and have made this innovation cycle possible. In particular, there has been a marked shift from the traditional “enterprise” stack (EJB2, .NET), and high ceremony development methods (RUP) to a more agile approach (XP), with a full embrace of more open tools and frameworks. Sometimes termed “lightweight” approach, this shift has included rapid evolution in web frameworks (Spring, OSGi) and the arrival of highly productive variations on these, most notably Ruby on Rails and its derivatives, usually bound to a relational database (MySQL).

Coupled with this evolving stack, web browser performance, stability and capability has improved dramatically. The client side web is now clearly formed by the nexus of HTML, CSS and JavaScript, with the latter in particular the lynchpin of significant innovation in the usability, power and flexibility of web applications. A set of JavaScript libraries (jQuery and others) has radically altered the usage patterns within the browser, unleashing unsuspected features in a robust environment. All three are sometimes grouped under the term HTML5, which in addition includes standardized approached to geo-location, 2D and 3D graphics, offline services, standard communications channels and more. Although not quite accurate, the term HTML5 usefully encapsulates the reach and ambition of the latest wave of browsers, and would seem to have significant momentum from all major players, hardware and software and infrastructure.

7.2 Revolution

There are signs, however, that this evolutionary approach may have run its course. The “lightweight” development stack of Relational Database, Component Service/Framework + Template Engine, all running on a Linux back end (a variant of the so called LAMP stack), is encountering a major shift in the underlying infrastructure - the arrival of the cloud. In particular, cloud based services coupled with advances in virtualization, have altered the principles around which applications have been architected to date. When this is also combined with various models for smart phone/tablet development, there is an argument that we are entering into another inflection point, comparable to the one foreseen in 2004.
What this particular movement will lead to is as yet unclear. However, it seems certain to yield new opportunities in services, mobility, flexibility and productivity in application development and deployment. In this context, there are signs of disruption within the current development stack. Although significant stability has been achieved since 2004, many its tenets are now being called into question:

- **Database**: The dominance of the relational database is not longer given. The “NoSql” movement is gathering pace with many open implementations of this broader, and perhaps more scalable architecture for the data store (MongoDB, CouchDB). When coupled with Googles Map/Reduce, it may be possible for more highly capable and intelligent systems can be constructed at a fraction of the cost for traditional relational systems.

- **Middleware**: Having already preceded though a series of major shifts over the past decade (rise and fall of Object Request Brokers, rise and fall of EJB, rise and stabilization of web frameworks), middleware is a useful touchstone when assessing the state of software and services. Evidence is mounting that the sheer complexity of current enterprise stack (J2EE, .NET) is causing profound limitations in the scale and reach of applications thus architected. The lightweight stack, evolved in some sense as an alternative to the traditional stack, may have reached its peak in Heroku, a marriage of cloud based services with a stable web framework. However, more disruptive technology is already emerging. In particular, the key to truly scalable services has always been the approach to concurrency. A radical alternative to traditional threading model (embodied in Heroku) is emerging. In particular, successive attempts to solve the concurrency problem (discussed below) are converging towards a more radical approach; namely the so-called non-blocking option.

- **Client**: The rise of the app store model is still taking shape. In particular, the introduction of this model to the general web (Google Chrome Web Store) may generate unforeseen consequences and trajectories in services and apps. For instance, the Chrome web store contains many applications that are indistinguishable from their apple app store equivalents (e.g. New York Times). However, these applications are full HTML5 (not native), are by definition more cloud oriented, and are thus liberated from highly restrictive (and complex) native app development toolkits.

### 7.3 Prediction

The Horizon 2020 (H2020) roadmap\(^\text{32}\) outlines a future world of predominantly mobile users accessing software based services through the cloud. This change in usage and user behaviours will require a flexible framework to facilitate the user. The H2020 document and the analysis presented here within this deliverable allows us to make a number of predictions. The mainstream programming language for the next ten years will be JavaScript. Once considered a toy language useful only for checking form fields on web pages, JavaScript will dominate enterprise software development. Why this language and why now? Today, the Java language is the one to beat. Java dominates enterprise software development. JavaScript and Java may have similar names, but the similarity ends there.

---

Though sometimes confused, they are very different languages. JavaScript owes its name to an accident of history; a failed and very strange marketing ploy from the early days of the web, when Netscape tried to leverage the growing popularity of Sun Microsystems’s new Java language. Both companies have now retired from the industry.

JavaScript is the language that web designers use to build web pages. However, it is not (yet) the language the software engineers use to build the business logic for those same web sites. JavaScript is small, runs on the client, the web browser. It’s easy to write unmaintainable spaghetti code in JavaScript. And yet, for all these flaws, JavaScript is the world’s most misunderstood language. Douglas Crockford, a senior engineer at Yahoo, is almost singlehanded responsible to rehabilitating the language. In a few short, seminal online essay published shortly after the turn of the century, Crockford explains that JavaScript is really LISP, the language of artificial intelligence. JavaScript borrows heavily from LISP, and is not really object-oriented at all. This curious design was well suited to a simple implementation running in a web browser. As an unintended consequence, these same mutations make JavaScript the perfect language for building cloud computing services.

The authors predict that within ten years, every major cloud service will be implemented in JavaScript, even those from Microsoft. JavaScript will be the essential item in every senior software engineer’s skill set. Not only will it be the premier language for corporate systems, JavaScript will also dominate mobile devices. Not just phones, but also tablets. All the while, JavaScript will continue to be the one and only language for developing complex interactive websites, completely drowning out old stalwarts such as flash, even for games. For the first time in a history, a truly homogeneous programming language infrastructure will develop, with the same toolkits and libraries used from the top to the bottom of the technology stack - JavaScript everywhere.
8 A Methodology for Architecting a Cloudlet

This section deals with implementation strategies widely adopted across a number of domains including cloud computing. As implementation strategies evolve, new uses not previously considered, but that are successfully deployed within disparate domains, might benefit the design of cloud based services. By investigating the best practices in domains such as Policy Based Network Management, Social Computing and traditional enterprise deployments, a greater understanding of what technologies and approaches for building a cloudlet and managing the supporting ecosystem can be arrived at. This section examines the current best practices for software engineering in the cloud. Included in this analysis is a comprehensive analysis on the software paradigms charting their evolution towards cloud computing deployment.

8.1 Design Patterns

Programming is a rich discipline and practical programming languages are usually quite comprehensive in breadth and terminology. Solving a programming problem requires choosing the right concepts, with knowledge about the structure and strengths of programming languages vital ([27]). All but the smallest problems require different sets of concepts to address aspects of the problem at hand. This is why programming languages should support many paradigms. A programming paradigm is an approach to programming a computer based on a mathematical theory or a coherent set of principles. Each paradigm supports a set of concepts that makes it the best for a certain kind of problem ([28]). Complementing a language paradigm is the concept of a design pattern, which is a general, repeatable solution to a commonly occurring problem. [29] presents a collection of popular design patterns tailored at solving the problems of object generation and interaction rather than a global solution. The patterns are broken down into three categories; 

- **Creational Patterns**, used to provide ways to instantiate objects or groups of objects;
- **Structural Patterns**, which define relationships among objects and
- **Behavioural Patterns**, defining manners of communication among objects. Systems of patterns interacting with each other helped achieve an overall solution architecture that was maintainable over time. The POSA (Pattern-Oriented Software Architecture) movement began with [30] and subsequently followed up with four more volumes (POSA 2-5) [33]. The volumes published represent a catalogue of design patterns that addressed core problems within software engineering, covering performance, availability and minimising risk to deployments. Developing software to pattern based specification provides a documentation of the system at a design level. In modern design methodologies, such as the Agile methodology ([35]) or the Extreme (XP) Methodology ([36]), documentation is key. The ability to interact with the end user

---

33 The POSA series of books:

- POSA2: Patterns for Concurrent and Networked Objects ([31])
- POSA3: Patterns for Resource Management ([32])
- POSA4: A Pattern Language for Distributed Computing ([33])
- POSA5: On Patterns and Pattern Languages ([34])
in an implementation independent manner is crucial for requirement gathering. The initial requirement gathering is supplemented by iterations of requirement meetings with the end user and responding to changes and requests in a timely manner. Design Patterns help in faster and more effective design for software, with reduction in effort achieved through making use of existing/standard patterns in design. Modern methodologies are suited to the current demands of shorter time frames and easily adaptable software ([37]). Marrying the two approaches has seen a greater understanding in how to produce sustainable software within tighter timeframes. This is particularly important when considering the fast change of pace that is modern cloud based computing. Interfacing with other technologies requires not only an extensible methodology but a flexible approach to produce compatible software in the shortest turnaround possible.

### 8.2 Language Paradigms

An understanding of such design patterns, methodologies and paradigms was matched at an implementation level. A case for object technology as a solution to many issues within software engineering was investigated as far back as Simula 1 ([38]), Simular 67 ([39]) and Smalltalk ([40]), culminating with the work of [41]. Object Oriented (OO) languages gained much traction as a result, with complementary design patterns ([29]) emerging. OO became one of the core reference paradigms due to the large uptake. The language style brought alternative mechanisms, including inheritance, object composition and polymorphism to bear on problems. This fresh perspective and innovative way of thinking was aided by the easier learning curve. Memory management (garbage collection), a means of dereferencing memory automatically, is a simple example of a complex mechanism now handled by the language ([42]). OO languages such as Java ([43]) gained widespread adoption. However, Java, among other languages, gained such popularity that it was inevitably used in scenarios that were outside the envisioned scope of the initial design. For example, [44] showed OO languages to be weak at modelling non-standard composition scenarios. However, the language evolved over time and addressed flaws and weaknesses ([45]) to remain a staple language in modern implementations.

The code execution component of Java is termed the Java Virtual Machine (JVM) with the specification freely available and documented ([46]). The JVM executes Java Byte code, an instruction set derived from the source language, which was primarily compiled Java programs. As Java Byte code was simply an instructional set, other languages availed of JVM compatibility by compiling down their source code into Java Byte code. Scala ([47]) and Groovy ([48]) are two such languages, although bindings exist for multiple languages [49]. JVM compatibility is attractive due to the portable nature of how the code is executed, leading to the term *write once, run anywhere* ([50]). Taking this approach, languages such as Groovy allow for direct Java authoring within Groovy classes. As the compiled output is Java Byte code, this is a valid approach. This allowed for a generation of programmers to adopt a new style, different in many respects to Java but allowing that safety net of pure Java coding to integrate with legacy systems. The JVM based languages that emerged brought with them a new set of features which programmers could avail of. Dynamic typing, where the type of a variable is not known until run time ([51]) and closures, where a safely scoped function can execute while retaining captured variable state ([52]), emerged with these languages.
ECMAScript is a standardised object-oriented programming language for performing computations and manipulating computational objects within a host environment ([53]). JavaScript (JS) is the more traditional name for ECMAScript, and JavaScript (JS) as a language is the most widely deployed programming language in history, with every browser ever developed containing a JavaScript interpreter. Initially regarded as a very limited language, its true nature and power had only recently been appreciated in any depth ([54]). Deployment on the server side has been attempted in the past, ([55]) but the true power of the language in that regard only emerged due to constructive thinking around the concept of concurrency and the challenge of handling mass amounts of simultaneous connections in cloud based infrastructure where concurrency has costs associated with it.

### 8.3 Concurrency

The challenge of concurrency is made concrete by what is known as the C10K problem, first posed by [56]. The C10K problem is this: how can you service 10,000 concurrent clients on one machine. The idea is that you have 10,000 web browsers, or 10,000 mobile phones all asking the same single machine to provide a bank balance or process an e-commerce transaction. The reverse of this problem is a similar challenge ([57]). While the number of connections is largely arbitrary, the problem itself is a classical concurrency problem which saw attempts at programmatic solutions emerge to provide safe access to resources. Threads, a lightweight process termed a thread of execution, popular in some languages such as Java ([58]), emerged as a solution. Threads evolved as technology became multiprocessor and parallel execution was sought, allowing multithreading, furthering the efforts to meet the C10K problem. In threaded systems, threads can share resources and memory but locks are associated with specific data structures. Emerging languages such as Scala took an alternative approach to memory management within threading, termed Actors ([59]), which avoids shared data structures and consequently any resource locking. However, both approaches can be termed Blocking Input/Output (I/O) as separate threads are created with their own stacks and program counters. The opposite of Blocking I/O, termed Non-Blocking I/O is achieved through extensive use of callbacks in API design and usage and has been shown to address this C10K problem. In this instance, any possible opportunity for blocking, be it a computational intensive task or accessing a resource, is replaced by passing a callback parameter to be invoked on completion of the deferred task or I/O request. JavaScript is a Non-Blocking language designed around callback execution and the development of high performance network programs ([60], [61], [62]) is now a possibility. The popularity and widespread availability of JavaScript execution environments, may see an effort akin to the Byte Code compatibility of the JVM begin to emerge. Already some languages such as Smalltalk\(^{34}\) and Java\(^{35}\) can run on top of the JavaScript Engine. While this cross compatibility among languages is attractive, languages such as CoffeeScript (CS) have been designed specifically around compilation to Javascript ([63]).

---

\(^{34}\) Amber and Squeak are implementations of the Smalltalk language that runs on top of the JavaScript runtime: http://amber-lang.net/ and http://squeak.org respectively

\(^{35}\) The Google Web Toolkit allows the authoring of JavaScript front end applications in Java: https://developers.google.com/web-toolkit/
8.4 Domain Specific Languages

A Domain Specific Language (DSL) is defined as a computer programming language of limited expressiveness focused on a particular domain ([64]). DSLs excel at taking certain narrow parts of programming and making them easier to understand and therefore quicker to write, quicker to modify, and less likely to breed bugs, overall increasing the productivity of the programmer. An additional benefit goes beyond programmers, to the field of domain experts or end users who can be exposed to the end code and make more informed input on the structure and design of the solution.

Common to all DSLs is that they do this by making the data structures and operations from their problem domain the basic building blocks of the language ([65]). DSLs can come in two variants, an internal DSL and an external DSL ([66]). An internal DSL is written and embedded within an existing host language. This can take the form of internal mini languages, where a subset of the overall language is used, or language enhancements, where DSL techniques through metaprogramming enhances the base language to make it look and feel like a DSL. The host language should provide a feature set to facilitate a DSL, the closure capability, for example, is a desirable feature. An external DSL has its own syntax with a full parser required to process the language, the freedom attained is at the cost of extensive development time. Valuable lessons were learned from initial DSL deployments and developments ([67]) and best practices established and documented ([68]). DSLs are much more mature with a support in the form of toolkits and design guidelines ([69]) available to end users.

8.5 Social Computing User Data Management

Social media networks have become hubs for our digital life. We share anything from pictures to personal information with our peers and the social network. Social networks such as Facebook represent the digital identity of people. They enable them to find each other through common data such as schools or interests. In order to be found by a person, the user needs to provide accurate data. This may also be required by the terms of service of the social network (e.g. Facebook). Users are motivated to provide their names, schools, birthdays or other personal data to the social network. They do this in order to use their services as intended (e.g. to be able to be found by a search). This results in an identity that represents the user digitally. It can be used to reference him, provide services to him or answer claims about him. Social networks have become identity providers that can be used by third party services to obtain user data. This requires authentication and authorization protocols.

- **OpenID** is an authentication protocol that enables third parties to grant access to a service based on their trust in an identity provider such as Google or Facebook. The user, when trying to use these services, is redirected to the identity provider in order to log in. After a successful login, the user is granted access to the service without the need to create and account. OpenID has been developed for authentication. It lacks to the ability to pass data between the service and the identity provider. However, passing data or claim-results (e.g. > 18 is true) to service providers is essential to reduce the fragmentation and repetitive input of user data. It enables service providers to customize or restrict services based on these information. The user's age for example
can be used to restrict access to a service. However, asking the user directly is not only a disruption, it is also easily circumvented. The attribute exchange (AX) extension has been developed to add such functionality to OpenID, but has not been widely adapted.

- **OAuth** has been developed as a standard, to authorize third parties and grant access to resources that would typically not be accessible by them. It was designed specifically to enable services to access user data without the need to provide a password to the services. Instead a token, which has an access scope and can be revoked, is used to grant access. For example, a third party service may request data about a user from the resource server. The authorization server will ask the user if such access should be granted, listing the attributes that are requested. If the access is granted, it will provide a token to the service. The service is now able to access or set the data on behalf of the user. The user does not have to repeatedly input the data needed by the service nor does he have to provide his password to the service. Once a resource server is required to hold validated data (e.g. a bank or a government agency) the service provider can be assured to obtain valid data as well. Such certified trusted providers can be used by services that require accurate information e.g. a payment service or for validating the users age. OAuth 1.0 has been deemed complicated to implement. It is error-prone due to the required signatures and struggled with insecure implementations. Major companies have been part of the development OAuth 2.0 and have adopted it. However, OAuth 2.0 has been developed as an extensible authorization framework which leaves many implementation details open to the implementing party. As such, it is likely that OAuth 2.0 implementations are not interoperable between parties.

- **OpenID Connect** is an identity layer on top of OAuth 2.0 that provides a RESTful API. The protocol allows service providers to verify the identity of a user and to obtain basic profile information about him. This basic profile may include the e-mail, name, date of birth or the gender. Therefore, OpenID Connect provides a minimal interoperability-base and adds authentication and single sign-on (SSO) functionality to OAuth 2.0.

In order to provide cloud based services with user data, platforms need to support authentication and authorization protocols. Social media networks have become such platforms. However, they keep their users on closed platforms and have no interest to interconnect their platforms (see OpenSocial). The users of one platform cannot freely connect with the users of another. They cannot migrate their digital identity to another provider. They are subject to different jurisdictions, depending on their platform. Service and application developers have to address each platform independently. There are no standards deployed by major platforms to support direct interoperability. One of the challenges to an open cloud service platform is to support as many use cases as possible via open and free standards. It is therefore necessary to look beyond the protocols which are currently deployed.

The OAuth 2.0 framework allows arbitrary access to resources through tokens. However in order to build interoperable user centric systems additional interoperability standards are needed.

---

36 http://occi-wg.org/
• **System for Cross-Domain Identity Management**\(^{37}\) \(^{38}\)(SCIM) is under development at the IETF. It is designed to manage user identities in cloud environments. It standardizes methods for creating, reading, searching, modifying and deleting user identities and related objects. The focus is the management of these resources across administrative domains with the goal to simplify common tasks related to user identity management. It also aims to provide a schema definition and discovery, bulk manipulation operations, a SAML binding and a mapping between the LDAP and the SCIM schema definitions. The SCIM Protocol is an application-level, REST protocol.

• **User Managed Access** (UMA) can be used to manage the access to distributed resources. It is designed to give a user's a unified control to authorizing who can access their personal data (e.g. email), their content (e.g. photos), and services (e.g. manipulating a shared document or photo). UMA also allows users to make demands towards the requester. If the demands are met, access to the resources is granted. This can be seen as a reversal of the terms of service procedure. Typically a user is required to agree to the demands of a service. Under UMA, services may be required to provide certificates or agree to the users terms of access (such as data protection assurances) before they can access the data. The requester can be held liable if an agreement is broken. This idea may lead to an infrastructure in which services react to the users demand and provide different features depending on the data they may access. The control over these resources is managed by an authorization manager the user controls. The server may protect any resource an UMA-enabled host holds. The host’s data is managed by the user as well. If a requesting party is accessing a resource, the authorization manager may grant the access depending on the users configuration. This approach differs from OAuth as it may allow access to resources without the user's direct involvement. For example, a user Alice has uploaded a video to UMATube. The video is a documentary that may shock younger audiences. Since UMATube is a UMA-enabled host, Alice protects the video with the demand to be over sixteen. A viewer Bob who is over sixteen finds the video via a search and is able to watch it as he assured UMATube he is old enough. Other users under the age of sixteen may not find the video via a search or may be blocked from accessing it. Unregistered users are handled by Alice’s policy as well. They need to personally provide their age in order to access the content. Such an access policy may be misused. However Alice feels is this case, the level of assurance is adequate enough to protect children from stumbling over the video.

• **Security Assertion Markup Language** (SAML) is a widely used enterprise standard to facilitate authentication, authorization and single-sign-on.

• **eXtensible Access Control Markup Language** (XACML) is a widely used enterprise standard as well which provides policy based access control. XACML has been considered by UMA for their policy based terms-setting ability while OpenID Connect is preferred to SAML.

• **Open Cloud Computing Interface**\(^{39}\) (OCCI) has been initially created as a remote management API for IaaS model, but has since evolved into a flexible API. It focusses on

---

\(^{37}\) [http://www.simplecloud.info/](http://www.simplecloud.info/)

\(^{38}\) [http://www.ietf.org/mail-archive/web/scim/current/msg00450.htm](http://www.ietf.org/mail-archive/web/scim/current/msg00450.htm)

Mobile Cloud Analysis

integration, portability, interoperability while being extensible. The current release of the Open Cloud Computing Interface supports multiple cloud categories.

- **Gluu’s OX**\(^3^9\) exists an early on cloud identity platform that integrates OpenID Connect, UMA, SCIM and others.

## 8.6 Policy Based Network Management

The management of interactions among entities within a network is often referred to as Policy Based Network Management (PBNM). This section deals specifically with access control within this domain, with respect to the access of services and content. PBNM is a condition, action, response mechanism, designed to provide an automated response to conditions within a network according to pre-defined rules encoded within policies. The definition put forward by [70], and supported by the wording within RFC 3198 ([71]), defines a Policy as a set of rules that are used to manage and control the changing and/or maintaining of the state of one or more managed objects. [71] also defines responsibilities for core elements that interact with Policies, namely the Policy Decision Point (PDP) and Policy Enforcement Point (PEP). The definitions put forward and used within this report, are as follows:

- **PDP**: The system entity that evaluates requests against applicable policies and renders an authorisation decision
- **PEP**: The system entity that performs access control, by routing requests and enforcing received authorisation decisions.

Policies are initially described in high level natural language and are capable of describing a range of activities ranging from predefined tasks, which must occur, performance related metrics, which may be protected by Service Level Agreements (SLA), and simple access control rules. Examples include:

- Allow the admin group write access to the membership database
- Run backups at midnight
- Network availability must be 99% over the course of the week

Understanding the variety of requests that may be encoded within policies and that may manifest itself as an access control request is important. The high level examples above are all variants of an access control policy.

### 8.6.1 XACML

The eXtensible Access Control Modelling Language (XACML) is an OASIS (Organisation for Advancement of Structured Information Standards) standardised specification for an access control
policy language and request language. XACML is implemented in the eXtensible Markup Language (XML), with an accompanying processing model to evaluate access control requests according to the rules contained within the policies ([72]). XACML extensively references [71] ensuring commonality with respect to definitions and terminology used within the world. XACML defines access control as controlling access in accordance with a policy returning an authorisation decision, typically permit, deny or Not Applicable, derived from evaluating the applicable policy with respect to the incoming request, as returned by the PDP to the PEP. This tightly defined sequence of events has ensured that XACML has become an industry standard for access control and the management of interactions, deployed within multiple domains.

8.6.2 Policy Authoring and Conflict

Mapping XACML closer to the source domain which needs to be controlled has potential benefits. The researchers in [73] deployed an abstract language on top of XACML in order to provide greater control over web service security. A user friendly language deployed on top of XACML provides a layer of abstraction on top of the heavy XML policies which can be difficult to read and interpret from an end user perspective. Attempts to make policy authoring more transparent and user friendly is a popular theme within XACML research. The price of simplifying the authoring is often conflicts appearing within the policies. A conflict, in policy terms, is when multiple policies are applicable to the same request, but result in different responses. XACML provides combining algorithms such as permit-overrides, where the first permit will override the overall decision, among other algorithms ([74]). The inbuilt combining algorithms can often be difficult to apply, particularly when XACML has grown out beyond its initial intention of a single domain environment. Federated domains and multiple layers of policies bring additional complications to detecting and resolving conflicts ([75]). A drawback of integrating ease of policy authoring mechanisms and the associated infrastructure required to ensure conformity, with respect to conflict analysis, is the size of the overall deployed system. The potential impact on the response times if policy conflict analysis has to run constantly can be profound in a high volume request environment. In an environment with a large number of policies, the time it takes to converge on a conflict can be in the order of minutes ([76]). The broader execution context, such as the impact on CPU and memory resources, is often ignored when evaluating policy authoring and conflict tools.

8.6.3 Policy Performance

The performance of a PDP is an important access control system requirement. It is relatively easy to scale out the stateless PEP functionality, but the stateful PDP function is difficult ([77]). Therefore having an efficient and performant PDP is a critical requirement, particularly in domains where a large volume of lightweight requests are issued. Making XACML policy evaluation more efficient is difficult from the point of view of the complex structures that are contained within XACML policies. [78] proposed a radical rethinking of XACML, converting policies and requests from XML to a numerical format. A supporting prototype PDP, capable of understanding the numerical format, showed dramatic performance improvements over the traditional PDP implementations that used XML. It is difficult however to abstract the performance gain from the representation format and the implementation logic behind the PDP. [79] examined the algorithms implemented within XACML
identifying problem scenarios which XACML currently struggles with. Additional solution algorithms were presented as a means to empower the PDP with logical choices that would represent a performance improvement when dealing with policy requests from the problem domains identified.

Bindings to XACML are a popular way of increasing evaluation and performance by making the PDPs more flexible and less reliant on external systems. [80] presented a means of translating REST (Representational State Transfer) based requests in order for a single PDP to be able to interpret requests from multiple PEPs. REST is a popular style of software architecture for distributed systems supporting web services ([81]). The work carried out largely centred around SOAP (Simple Object Access Protocol) ([82]) based PDPs receiving non SOAP queries which traditionally would have been an interoperability issue requiring an external translation, pushing out the overall request-response time. Securing communication with group environments ([83]) has also been explored within XACML. Available research in this domain seems to focus on the surrounding architecture in low volume transactions. The performance profile at larger levels of scale in a traditional social network are quiet varied. Access control has not been deployed within this domain largely due to a lack of privacy protection when dealing with third party applications and a lack of interoperability among different Access Control Policy Languages ([84]). Considerations for response times within such an environment have not been explored extensively as a consequence.

8.6.4 Policy Representation
Policy representation within XACML, by its specification, was bounded to XML. The extensible nature of the specification allowed for the development of XACML within diverse fields. The authors of [85] embedded access control policies within digital content using a XACML like document. This approach encoded the item to be protected with the protecting policy. By encoding both resource and policy information within one single XACML document, the PDP no longer requires a large scale search of a policy store, as all relevant information is at hand. The research carried out implemented a number of prototypes, largely focusing on the authoring of the single XACML document. While no performance characteristics were recorded, the approach showed that a rethink on the structure of a request and policy could bring performance benefits from a PDP point of view.

[78] took a numerical approach to policy representation as a means of policy normalisation. The idea of policy normalisation is to develop a common policy language or normal form to represent policies from any application. The major benefits put forward include a normal form that has a simple logical structure and the reuse of policy evaluation algorithms. The resulting representation however is opaque, non-symbolic and difficult to analyse. Presenting such a representation to domain experts and XACML experts alike would be difficult, despite the performance benefits presented in the research.

[86] examined alternative data representation formats, examining the effect of policy representation on PDP performance. The XML based policies were hand translated to JSON capturing the semantics of the XACML policy without the bloat of metadata that accompanies XML. The resulting performance improvement was an 8 times improvement on the current state of the art PDP implementations. An
alternative experiment was carried out where the inputted XML policies and requests were translated at run time, showing that PDP evaluations still struggle with inefficient formats. This research highlighted the need for minimal friction between PDP design and policy representation. The experiments were described in further detail in [87] and added to with a CoffeeScript representation. The authors note that traditional data representation formats may not be sufficient to encode human centric policies, particularly for access control and privacy. The resulting CoffeeScript PDP and accompanying policies reported speeds of an order of magnitude faster than the traditional XACML implementations.

8.6.5 PBNM Summary

Strong management principles have relied extensively on classical networking approaches, with a human centric mechanism a secondary requirement to securing the network. The rulebases are difficult to interpret and establish, with administration often limited to a number of individuals. XACML, one of the most successful access control management platforms for management interactions is currently being deployed in domains outside of the scope of the original intention of the specification. The heavyweight nature of the design and the all-encompassing specification, which must be met in its entirety, to ensure conformance, can have a negative impact on the performance and scalability of a system. Despite the drawbacks, which have been addressed in part through research attempts highlighted within this section, PBNM approaches such as XACML, strongly define management flows and access to data. The design and implementation of a cloudlet can learn from the style of management as put forward by such a system. An approach that can bring XACML principles into the management of user cloudlets, in a performant manner, is an attractive combination.
9 Conclusion and Recommendations

This report has shown the evolution of best practices with respect to design of applications designed to be consumed in a modern, mobile oriented world. The usage of the Cloud from the point of view of the core stakeholders was also outlined. An innovative approach to managing users data management and their interactions within cloud based services was specified as a Cloudlet. Accompanying this specification was a comprehensive overview of the current best practices for designing cloud based applications and platforms. From this, the authors would like to propose the following recommendations for the design and implementation of a Cloudlet and supporting platform:

- **Concurrency**: Diverse approaches to programmatically coping with concurrency have long been a source of contention among software developers. This report recommends that within the context of cloud computing, a Non-Blocking I/O approach to concurrency can facilitate more responsive and scalable platforms.

- **Underlying Platform**: With the Node.js framework, JavaScript is no longer just a language supporting user interaction within browsers. Based on the Google initiated, open source V8 JavaScript engine, JavaScript, or languages that can compile down into JavaScript, can be compiled into highly optimised server-side machine code on the y. The Non-Blocking nature of JavaScript is present within Node.js with all requests gradually executed in sequence through the usage of callbacks. Node.js allows an elegant solution to be engineered for traditional scalability problems and an alternative approach for domains that might benefit from a Non-Blocking I/O approach.

- **Design**: A modular component based design has been shown to provide more flexibility in adapting to the fast pace of change of modern cloud based services. Analysis of exiting frameworks has shown that a component style to building the platform can allow for greater extensibility and allow the cloudlet platform interact with existing cloud based providers seamlessly.

- **Access Control Mechanisms**: This report recommends that strong access control mechanisms based on the PBNM domain should be deployed within a cloud based solution. Such a deployment can be heavyweight and counterproductive in the cloud, however recent advances in policy authoring and representation might allow a scalable performant deployment cloud side. These strong management principles are required to ensure confidence in the end user and allow for a more robust access control solution.
Annex I: References


Last accessed: 17/07/2012.

Last accessed: 16/07/2012.


[16] Facebook, 2012. Facebook Graph API Specification


http://www.intomobile.com/2012/04/21/foursquare-surpasses-20-million-users-2-billion-checkins/
Last accessed: 16/07/2012.


[29] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994.


Last accessed: 16/07/2012.


[64] M. J. Fowler, *Domain-Specific Languages (Addison-Wesley Signature Series (Fowler))*. Addison-Wesley Professional, 2010.


