Open Group Guide

Cloud Computing Portability and Interoperability

Guide

Cloud Computing Portability and Interoperability

ISBN: 1-937218-30-0

Document Number: G135

Published by The Open Group, April 2013.

Comments relating to the material contained in this document may be submitted to:

The Open Group, Apex Plaza, Forbury Road, Reading, Berkshire, RG1 1AX, United Kingdom

or by electronic mail to:

ogspecs@opengroup.org

Contents

Preface

The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through IT standards. With more than 400 member organizations, The Open Group has a diverse membership that spans all sectors of the IT community – customers, systems and solutions suppliers, tool vendors, integrators, and consultants, as well as academics and researchers – to:

- Capture, understand, and address current and emerging requirements, and establish policies and share best practices
- Facilitate interoperability, develop consensus, and evolve and integrate specifications and open source technologies
- Offer a comprehensive set of services to enhance the operational efficiency of consortia
- Operate the industry's premier certification service

Further information on The Open Group is available at www.opengroup.org.

The Open Group publishes a wide range of technical documentation, most of which is focused on development of Open Group Standards and Guides, but which also includes white papers, technical studies, certification and testing documentation, and business titles. Full details and a catalog are available at www.opengroup.org/bookstore.

Readers should note that updates – in the form of Corrigenda – may apply to any publication. This information is published at www.opengroup.org/corrigenda.

This Document

This document is The Open Group Guide to Cloud Computing Portability and Interoperability. It has been developed and approved by The Open Group.

The Guide explains the major portability and interoperability issues that arise when cloud computing is used. It makes recommendations to customers on how best to achieve portability and interoperability when working with current cloud products and services, and it makes recommendations to suppliers and standards bodies on how standards and best practice should evolve to enable greater portability and interoperability in future.

It covers all forms of cloud computing, as described in the NIST Cloud Computing definition [NIST DEFINITION].

The intended audience includes:

- Architect practitioners designing or using cloud computing solutions

- Vendors and providers of cloud computing seeking guidance on interoperating issues and practices
- Customers and buyers of cloud computing services seeking guidance on interoperability issues and practices
- Governments, non-governmental organizations (NGOs), and commercial industry bodies that have specific vertical industry or cross-industry issues and challenges and seek guidance on cloud computing interoperability
- Standards organizations and other bodies seeking a point of view from The Open Group on cloud computing portability and interoperability

Trademarks

Acknowledgements

The Open Group gratefully acknowledges the contribution of the following people in the development of this document, and in particular the leadership of Kapil Bakshi and Mark Skilton as project co-chairs.

- Omkhar Arasaratnam, IBM
- Kapil Bakshi, Cisco
- Larry Beser, Cisco
- Stuart Boardman, KPN
- Alex Brojba-Micu, KPN
- Rance DeLong, LynuxWorks
- Penelope Gordon, 1Plug
- Chris Harding, The Open Group
- Ed Harrington, Architecting the Enterprise
- Louise Hemond-Wilson, IBM
- Dave Hornford, Conexiam
- Shrikanth Kashyap, Wipro
- Krishnamurthy Krithivasan, Ernst & Young
- Parag Mehta, Cisco
- Tuomas Nurmela, Tieto
- Sreeparna Pal, TCS
- Bala Peddigari, TCS
- Gilbert Pilz, Oracle
- Jeffrey Raugh, HP
- Ron Schuldt, UDEF-IT
- Dipanjan Sengupta, Cognizant
- Tharmananthar Shankaradhas, Microsoft
- Dave Shepherd, EMC

- Palle Simonsen, Devoteam
- Mark Skilton, Capgemini
- Goran Strangmark, HP
- Hendrik Van Geel, IBM
- Martial Van Neste, CGI
- TJ Virdi, Boeing
- Christopher Voigt, IBM

Referenced Documents

The following documents and web sites are referenced in this Guide:

[.NET] The Microsoft .NET Framework; refer to: www.microsoft.com/net.

[ANDROID] The Android Mobile Platform; refer to: www.android.com.

[C] ISO/IEC 9899 – Programming Languages – C; refer to: www.iso.org.

[CAMP] Cloud Application Management for Platforms (CAMP); refer to: www.oasis-open.org/committees/tc_home.php?wg_abbrev=camp.

[CAP] Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services, Seth Gilbert and Nancy Lynch, ACM SIGACT News, Volume 33 Issue 2 (2002), pp.51-59; refer to: http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf.

[CDMI] Cloud Data Management Interface (CDMI), Storage Networking Industry Association; refer to: www.snia.org/cdmi.

[CIM] The Common Information Model (CIM), Distributed Management Task Force; refer to: dmtf.org/standards/cim.

[CLOUD GUIDE] Guide to Cloud Computing for Business, G114, August 2011, published by The Open Group; refer to: www.opengroup.org/bookstore/catalog/g114.htm.

[CMIP] ITU-T Recommendation X.711: ISO/IEC 9596-1:1998: Information Technology – Open Systems Interconnection – Common Management Information Protocol: Specification; refer to: www.itu.int/rec/T-REC-X.711-199710-I/en.

[CMIS] ITU-T Recommendation X.710: ISO/IEC 9596-1:1998: Information Technology – Open Systems Interconnection – Common Management Information Service; refer to: www.itu.int/rec/T-REC-X.710-199710-I/en.

[CSA] Cloud Security Alliance; refer to: cloudsecurityalliance.org.

[DMTF] Distributed Management Task Force; refer to: www.dmtf.org.

[ECMA262] ECMAScript Language Specification, ECMA; refer to: www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf.

[FIELDING] Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvive, 2000; refer to: www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[HTML] HyperText Markup Language, World-Wide Web Consortium; refer to: www.w3.org/MarkUp.

[HTML5] HTML5 Candidate Recommendation, World-Wide Web Consortium; refer to: www.w3.org/TR/2012/CR-html5-20121217.

[HTTP] Hypertext Transfer Protocol, Internet Engineering Task Force; refer to: www.w3.org/Protocols/rfc2616/rfc2616.html (see also HTTP over TLS: https://tools.ietf.org/html/rfc2818).

[IOS] iOS Mobile Platform; refer to: www.apple.com/ios.

[IP] Internet Protocol, IETF RFC 791; refer to: http://tools.ietf.org/html/rfc791.

[J2EE] Java 2 Platform, Enterprise Edition; refer to: www.oracle.com/technetwork/java/javaee.

[JAVA] Java Programming Language; refer to: www.java.com.

[JAVASCRIPT] Javascript Web Scripting Language; refer to (for example): www.w3schools.com/js/default.asp.

[JSON] Javascript Object Notation; refer to: www.json.org.

[LINUX] Linux Operating System; refer to: www.linuxfoundation.org.

[MSWIN] Microsoft Windows; refer to: http://windows.microsoft.com.

[NIST DEFINITION] Definition of Cloud Computing, US National Institute of Science and Technology, 2011; refer to: www.nist.gov.

[OAUTH] Open Authentication Protocol; refer to: http://oauth.net.

[OCCI] Open Cloud Computing Interface, Open Grid Forum; refer to: http://occi-wg.org.

[ODCA] Open Data Center Alliance; refer to: www.opendatacenteralliance.org.

[OGBOOKS] The Open Group Bookstore; refer to: www.opengroup.org/bookstore/catalog.

[OPENSTACK] Openstack Foundation; refer to: www.openstack.org.

[OSIMM] The Open Group Service Integration Maturity Model (OSIMM), Version 2, Technical Standard, C117, November 2011, published by The Open Group; refer to: www.opengroup.org/bookstore/catalog/c117.htm.

[OTA] Open Travel Alliance; refer to: www.opentravel.org.

[OVF] Open Virtualization Format, DMTF; refer to: www.dmtf.org/standards/ovf.

[OWASP] OWASP Top Ten Project; refer to: www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[OWL] Web Ontology Language, World-Wide Web Consortium; refer to: www.w3.org/TR/owl-overview.

[PHP] PhP Web Scripting Language; refer to: http://php.net.

[PYTHON] Python Programming Language; refer to: www.python.org.

[RDF] Resource Description Framework, World-Wide Web Consortium; refer to: www.w3.org/TR/2004/REC-rdf-primer-20040210.

[SLASOI] The SLA@SOI Consortium; refer to: http://sla-at-soi.eu .

[SOA] The Open Group definition of SOA; refer to: www.opengroup.org/soa/source-book/soa/soa.htm.

[SOAP] Simple Object Access Protocol, World-Wide Web Consortium; refer to: www.w3.org/TR/soap.

[SOARA] SOA Reference Architecture, Technical Standard, C119, December 2011, published by The Open Group; refer to: www.opengroup.org/bookstore/catalog/c119.htm.

[SOCK] The Sockets Programming Interface; refer to: http://pubs.opengroup.org/onlinepubs/7908799/xnsix.html.

[SNMP] Simple Network Management Protocol (SNMP), IETF; refer to: www.ietf.org/rfc/rfc3411.txt.

[SQL] ISO/IEC 9075:2011: Information Technology – Database Languages – SQL; refer to: www.iso.org.

[TCP] Transmission Control Protocol, IETF RFC 793; refer to: http://tools.ietf.org/html/rfc793.

[TOGAF] The TOGAF® Standard, published by The Open Group; refer to: www.opengroup.org/togaf.

[TOSCA] Topology and Orchestration Specification for Cloud Applications, OASIS; refer to: www.oasis-open.org/committees/tosca.

[UDDI] Universal Description Discovery and Integration Standard, OASIS; refer to: uddi.xml.org.

[UDEF] Universal Data Element Framework (UDEF), The Open Group; refer to: www.opengroup.org/udef.

[UNIX] The UNIX® Operating System; refer to: www.unix.org.

[VMAN] Virtualization Management Standards, DMTF; refer to: dmtf.org/standards/vman.

[WADL] Web Application Description Language, submission to the World-Wide Web Consortium; refer to: www.w3.org/Submission/wadl.

[WEF-1] Exploring the Future of Cloud Computing, World Economic Forum, May 2010; refer to: www.weforum.org.

[WEF-2] Advancing Cloud Computing: What to Do Now? World Economic Forum, 2011; refer to: www.weforum.org.

[WS-I] Web Service Interoperability Standards, OASIS; refer to: www.ws-i.org.

[WSDL] Web Service Description Language, World-Wide Web Consortium; refer to: www.w3.org/TR/wsdl.

[XML] Extensible Markup Language, World-Wide Web Consortium; refer to: www.w3.org/XML.

[XPath] XML Path Language, World-Wide Web Consortium; refer to: www.w3.org/TR/xpath.

1 Introduction

Cloud computing underpins an important part of economic activity today, and has the potential to make a major contribution to future growth.

To deliver its anticipated benefits, it must be easy to use and cost-efficient. This means that enterprises and individuals must be able to use cloud products and services as far as possible "off the shelf". The products and services should work together, and minimal effort should be needed to incorporate them into a user's systems.

This is the case for the basic Internet, but not yet for the additional components that constitute the cloud. Lack of portability of and interoperability between these components could mean that the potential of cloud computing is not fulfilled.

This guide analyzes cloud computing portability and interoperability. It makes recommendations to customers on how best to achieve portability and interoperability when working with current cloud products and services. It makes recommendations to suppliers and standards bodies on how standards and best practice should evolve to enable greater portability and interoperability in the future.

Following this summary, the Guide starts with two introductory chapters. Problem Statement (Chapter 2) discusses the problem, and Cloud Computing (Chapter 3) explains the basic concepts of cloud computing. The analysis then proceeds by distinguishing the important categories of Cloud Portability and Interoperability (Chapter 4).

The central part of the analysis is the definition of the Distributed Computing Reference Model (Chapter 5). Portability and interoperability reduce cost and effort because they allow enterprises to use products and services that work together "off the shelf". The model identifies the major components of cloud-enabled solutions that can be implemented by such products and services.

A high level of portability and interoperability, such as we have for the Internet, is achieved when products and services conform to interface standards. The Guide describes the Portability and Interoperability Interfaces (Chapter 6), and reviews the state of development of their standards.

Interface standards can apply to generic components, such as platforms and infrastructure, but applications have interfaces that reflect their particular functionality, and are not appropriate for standardization. The Guide puts forward a set of Application Design Principles (Chapter 7) to minimize their cost of integration.

The final chapter of the Guide contains Conclusions and Recommendations (Chapter 8) for how enterprises should proceed in the light of the current state of standardization, and for how standards bodies and the industry should work to develop standards and best practices that will improve portability and interoperability for the future.

Cloud Computing and SOA (Appendix A) explains the relation of The Open Group SOA Reference Architecture [SOARA] to the Distributed Computing Reference Model.

Example Use-Cases for WS-I and Raw HTTP (Appendix B) contains two small example use-cases illustrating the differences between the two main web service interface styles: WS-I and Raw HTTP.

The final section is a list of acronyms used in this Guide.

2 Problem Statement

2.1 Why are Cloud Interoperability and Portability Important?

Cloud computing underpins an important part of economic activity today. It has the potential to enable an increasing amount of such activity, and make a significant contribution to the growth of world Gross Domestic Product (GDP).

In the Spring of 2010, the World Economic Forum published a report that evaluated the impact of cloud computing technologies and highlighted the large potential benefits of adoption, ranging from economic growth and potentially sizeable improvements in employment to enabling innovation and collaboration [WEF-1]. While recognizing the many benefits of cloud, the report also highlighted very significant issues that could limit our ability to benefit from cloud computing. A second report, published in 2011 [WEF-2], identified eight areas in which governments and industry can take action to accelerate the deployment and adoption of public cloud technologies. Two of these are data portability and interoperability.

On data portability, the report says that:

"The fear of vendor lock-in holds back many potential users of cloud, while many government stakeholders are concerned about maintaining competitiveness in the cloud market. These concerns are lessened if it becomes quicker, easier, and cheaper for users to move data, and perhaps applications, between different cloud providers and between user premises and the cloud."

While data portability is identified as the primary concern, these words apply equally to applications portability, which the report also recognizes as an issue.

On interoperability, the report makes a call for action:

"Industry players should pursue the evolution of cloud offerings with the goal of facilitating interoperability among multiple (private and public) clouds. This will accelerate the growth of the overall cloud ecosystem."

Cloud portability and interoperability are not just matters of technical concern. They are questions of economic importance. Governments, and bodies such as the World Economic Forum, recognize the potential of cloud computing, and they understand that lack of portability and interoperability can prevent this potential being fulfilled.

2.1.1 Standard Components

For cloud computing to deliver the anticipated benefits, it must be easy to use and cost-efficient. This means that enterprises and individuals must be able to use cloud products and services as far as possible "off the shelf". The products and services should work together, and minimal effort should be needed to incorporate them into a user's systems. It should be possible to write

any user-specific software that may be required so that it is based on commonly available components that can easily be sourced from multiple suppliers.

Portability and interoperability standards enable the development of such products and services, and of user-specific software that works with them. The availability of these products and services in a free market will stimulate the growth of cloud computing and of businesses that use it.

2.1.2 The Internet

The Internet provides an excellent example of how portability and interoperability can enable economic growth. It is a hugely valuable world resource, supporting a massive amount of commercial and other activity. It has a few basic, well-defined components (host, router, network, etc.) that have clearly defined interface standards, such as the TCP and IP communications protocols [TCP, IP] and the sockets programming interface [SOCK]. Companies can procure products that conform to these interfaces easily, and knowing that they will work together "out of the box". It has even reached the point that members of the general public, with no specialist knowledge, can do this. The vast and growing use of the Internet, and the consequent benefits, are largely due to this high level of portability and interoperability.

The Internet is one of the foundations of cloud computing, and the current growth in cloud computing is partly due to the portability and interoperability that comes from conforming to the Internet standards. But cloud computing is more than the Internet: it provides information processing and storage, as well as communication and display. It has a different set of basic components to deliver this functionality, and the interfaces between these components do not have universally accepted, clearly defined standards.

2.1.3 Cloud Computing

Cloud computing has the potential to support commercial and other activity to an even greater extent than the Internet. The economic and social benefits that could flow from this are truly enormous. But this potential will not be realized without portability and interoperability. They are important – some would say vital – because of their ability to enable this technology-based revolution.

2.2 Why they are Difficult to Achieve

Portability of and interoperability between system components requires standards that define how the components behave. These standards must cover not only what the components do, but also some aspects of how they do it. In particular, they must allow reasonable speed of execution and security.

2.2.1 Creation of Standards

The creation of such standards is hard, for three main reasons. First, vendors may see their immediate commercial advantage in discouraging their customers from using products supplied by other vendors. They may therefore decide not to support applicable standards. Second, the detailed definition of effective standards requires agreement between producers on specific details of the interfaces. This may be hard to achieve, even when the producers have the desire to

achieve it. Finally, the identification and description of the requirements that the standards place on products or services requires skill, judgment, and experience of the subject matter. Poor standards, defined without bringing these qualities to bear, place a burden on implementers and users, without delivering real benefits. Unfortunately, poor standards are not uncommon.

The technical parameters that must be addressed in defining a good standard are quite complex. For cloud computing, and many other areas of IT, they include not only functionality but also aspects of the quality of delivery of that functionality. Two of these qualities are particularly important for cloud computing portability and interoperability: *security* and *performance*. Because they are so important, the description of the Distributed Computing Reference Model (Chapter 5) contains dedicated sections on them.

2.2.2 Difference Between Technology and Application Components

Many types of cloud computing component can have simple interfaces that can be defined in standards to which all instances can conform. This is, however, not the case for applications. Each application is different. There is reason to standardize the interfaces to some applications to enable collaboration in particular industry sectors, but otherwise it is desirable to allow variation, so that the interfaces can reflect specific product functionality, and individual vendors are free to introduce the functionality that they believe meets the needs of their customers.

Interoperability between applications "out of the box" is therefore not generally possible. Integration of applications can, however, be more or less easy, depending on how they are designed. An important aspect of cloud interoperability is that integration of applications should be as easy as possible. Addressing this issue is a major theme of this Guide.

2.2.3 Application Design Principles

Kirk Avery, Technical Working Group Vice-Chair of the Future Airborne Capability Environment (FACE) Consortium, made a very interesting remark in his presentation to The Open Group conference in Washington in July 2012. He said:

"One of the key factors inhibiting portability and interoperability of software components today is tight coupling."

Portability and interoperability need technical standards defining interfaces, but they also need something more: simple principles for the arrangement of the components.

The arrangement of Internet components is well understood, and the components are in fact loosely coupled. Standardization of cloud platform, management, and marketplace interfaces, as recommended by this Guide, will bring about a similar situation for cloud computing except, for the reasons given above, at the application level. The aim at this level should be to minimize the effort required to integrate components with each other.

This can be achieved if developers of cloud applications and application services follow principles such as loose coupling. The chapter of this Guide on Application Design Principles (Chapter 7) puts forward a set of principles to minimize the cost of integration for cloud computing applications and application services.

2.3 Architecture for Interoperability and Portability

Ecosystems and Enterprise Architectures enable interoperability and portability by defining building blocks with standard interfaces that can be realized by products and services that will work together without the need for a large amount of integration effort. Many of these products and services will be available commercially, "off the shelf", from multiple vendors. Those that are developed specially will interwork and can be used with standard components.

2.3.1 Architecture Domains

These architectures are often considered to have three domains, as in the TOGAF standard [TOGAF], from which the following definitions are taken:

- **Business Architecture**: A description of the structure and interaction between the business strategy, organization, functions, business processes, and information needs.
- **Information Systems Architectures**: The information systems architectures are Data Architecture and Application Architecture. Data Architecture is a description of the structure and interaction of the enterprise's major types and sources of data, logical data assets, physical data assets, and data management resources. Application Architecture is a description of the structure and interaction of the applications as groups of capabilities that provide key business functions and manage the data assets.
- **Technology Architecture**: A description of the structure and interaction of the platform services, and logical and physical technology components.

Cloud computing impacts on all of them.

2.3.2 Business Architecture

Cloud computing:

- Encourages enterprises to focus on their strengths, by making it easier to obtain non-core capabilities by purchasing external services
- Facilitates collaborative working between enterprises and between parts of an enterprise
- Enables faster development of new products and services, with reduced time to market.

2.3.3 Information Systems Architectures

Cloud computing brings new dimensions to Data Architecture, by introducing cloud data storage services, and by facilitating processing of "big data". Cloud data storage services include services based on new storage paradigms, including those characterized as "NoSQL", particularly the simple key-value pair paradigm. *Big data* typically means a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools. Social networking and other cloud services form an important source of such data, and cloud resources lend themselves to bursts of computationally-intense processing to obtain results from it.

To take full advantage of cloud services, an enterprise needs an architecture based on loosely-coupled services, not on information silo applications with tightly-coupled components. This Guide contains a set of Application Design Principles (in Chapter 7) that will assist the development of Enterprise Architectures that use cloud services to full advantage.

2.3.4 Technology Architecture

Cloud computing introduces new platform services and technology components, leading to Technology Architectures that are quite different from those that support traditional applications. This Guide contains a Distributed Computing Reference Model (Chapter 5) with Portability and Interoperability Interfaces (Chapter 6) to assist the development of Technology Architectures that incorporate cloud components and support cloud-based applications and data.

2.4 Performance and Security

Components must not only behave functionally in the cloud exactly as they do in in-house systems. They must also perform equally well, with equally good execution and response times, and they must be as secure in the cloud as they are in-house.

2.5 Legal Jurisdictions and Sovereignty

Large cloud suppliers operate multi-nationally. An enterprise using their services is likely to have processing performed in, and data moved between, different jurisdictions. This can place constraints on the processing that can be performed, on the movement of the data, and on the degree of control that the enterprise has.

Cloud computing can, however, be seen in this context as an opportunity, rather than a problem. For example, the growth of cloud computing could contribute to achieving the objectives of the European Union for improved interoperability of IT systems across national borders.

3 Cloud Computing

Cloud computing is a major development in information technology, comparable in importance with the mainframe, the minicomputer, the microprocessor, and the Internet. This chapter describes what makes it special.

3.1 The Cloud Concept

The essential concept of cloud computing is that IT resources are made available, within an environment that enables them to be used, via a communications network, as a service, as shown in Cloud Service (Figure 1).

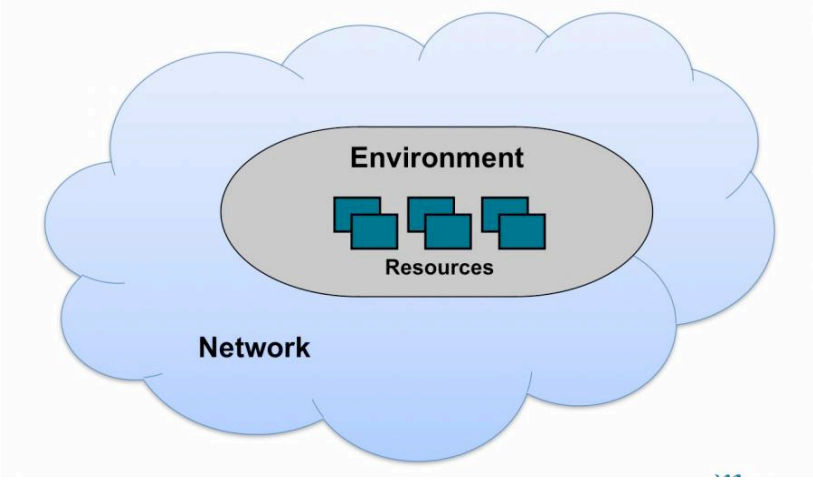

Figure 1: Cloud Service

The resources can be raw computation resources, such as processing power and data storage, or computation resources that are programmed to perform a function, such as software development, web conferencing, or CRM.

The raw computation resources are real, but the environment often contains a virtualization layer, so that they are presented as virtual resources.

The environment enables managers to select, configure, and deploy the resources. For raw computation resources, the environment might, for example, enable a manager to select a virtual processor of a particular power, attach a chosen amount of virtual disc storage to it, and make it accessible at a particular IP address. For other resources the environment might enable a manager to define the number of users and the types of user that can use the resource.

The communications network is typically the public Internet or a private network that uses the Internet protocols (an intranet). The Internet protocols are the foundation of cloud interoperability and portability. Other kinds of network will not be considered in this Guide.

The Internet, and communications networks generally, are often represented by cloud shapes in figures and diagrams. This is what has given cloud computing its name.

3.2 The NIST Definition

The US National Institute of Standards and Technology (NIST) has developed a definition of cloud computing that is widely accepted within the industry, and is adopted by this Guide [NIST DEFINITION]. It includes:

- A description of the essential characteristics that a cloud service should have
- A primary classification of the ways in which a cloud service can be provided – its deployment models
- A primary classification of cloud services – its cloud service models

3.3 Essential Characteristics

A cloud service should have five essential characteristics:

- **On-demand Self-service**: Managers can obtain services as and when needed, without human intervention.
- **Broad Network Access**: Users can access services using a broad range of network client devices, such as browsers, mobile phones, tablets, laptops, and workstations.
- **Resource Pooling**: Resources are not dedicated to particular consumers but are pooled and allocated as required.
- **Rapid Elasticity**: The amount of resource allocated to a consumer can be expanded and contracted easily and quickly to meet demand.
- **Measured Service**: The amount of resource consumed is measured, to enable automatic expansion and contraction, and to give visibility of usage to providers and consumers.

3.4 Deployment Models

A cloud service may be deployed:

- As public cloud, by a cloud provider enterprise for use, often on commercial terms, by user enterprises
- As private cloud, by an enterprise for its own use
- As community cloud, by a collaborating group of enterprises for use by members of the group

- As hybrid cloud, where the user enterprise deploys a composite service whose components are a mixture of public, private, and community cloud services

3.5 Service Models

Three cloud computing service models are distinguished in the NIST definition, and they are a good basis for an analysis of portability and interoperability issues. They are Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

- For SaaS, the resources that are made available are software application programs, and the environment enables them to be configured and deployed.
- For PaaS, the resources are software application platforms, and the environment enables them to be configured and deployed, and enables applications to be created, configured, and deployed upon them.
- For IaaS, the resources are virtual processors and data stores, and the environment enables them to be configured and deployed, and enables platforms and applications to be configured and deployed upon them.

A further cloud service model is often included, though it is not part of the NIST definition: Business Process as a Service (BPaaS). In this model, the resources that are made available are systems that include people as well as IT components. They perform complete business functions, such as payment processing. They are available over the Internet, or an enterprise intranet, and exhibit the five essential cloud service characteristics.

4 Cloud Portability and Interoperability

Portability and interoperability relate to the ability to build systems from re-usable components that will work together "out of the box".

A particular concern for cloud computing is *cloud on-boarding* – the deployment or migration of systems to a cloud service or set of cloud services. A common scenario is that some components cannot be moved to the cloud; for example, because of requirements for the enterprise to have complete control over personal data. On-boarding requires portability of those components that can be moved to the cloud, and interoperability with them of components that remain on in-house systems.

4.1 The Important Categories of Cloud Computing Portability and Interoperability

A system that involves cloud computing typically includes data, application, platform, and infrastructure components, where:

- *Data* is the machine-processable representation of information, held in computer storage.
- *Applications* are software programs that perform functions related to business problems.
- *Platforms* are programs that support the applications and perform generic functions that are not business-related.
- *Infrastructure* is a collection of physical computation, storage, and communication resources.

The application, platform, and infrastructure components can be as in traditional enterprise computing, or they can be cloud resources that are (respectively) software application programs (SaaS), software application platforms (PaaS), and virtual processors and data stores (IaaS).

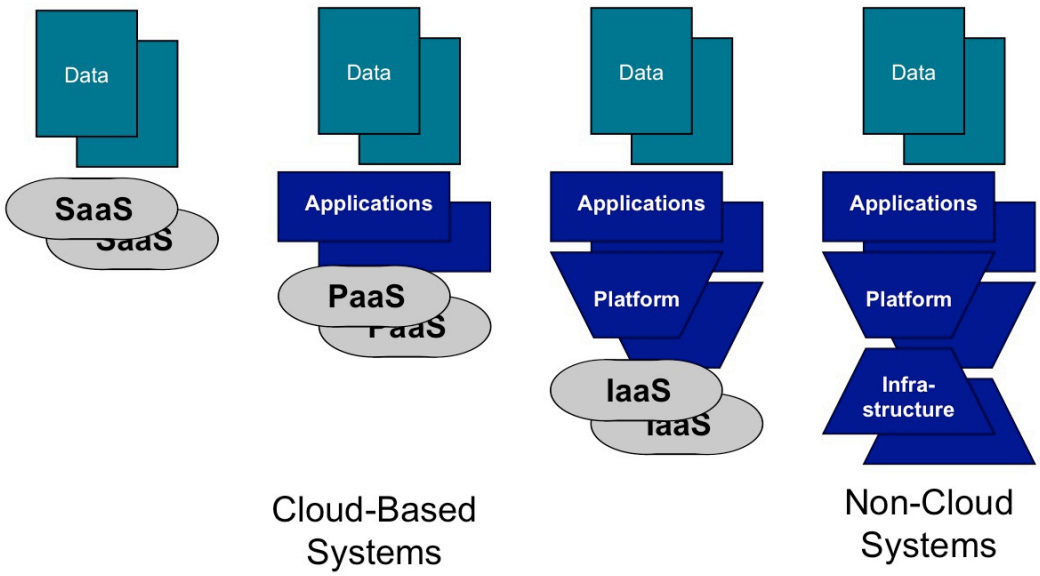

Figure 2: Data, Applications, Platforms, and Infrastructure

Non-cloud systems include mainframes, minicomputers, personal computers, and mobile devices owned and used by enterprises and individuals.

Data components interoperate via application components rather than directly. There are no "data interoperability" interfaces.

Portability and interoperability of infrastructure components are achieved by hardware and virtualization architectures. The interfaces are mostly internal to the IaaS and infrastructure components shown in Data, Applications, Platforms, and Infrastructure (Figure 2). The interfaces exposed by these components are physical communications interfaces: these are important, but are the same as for traditional computing. For these reasons, infrastructure portability and interoperability are not discussed further in this Guide.

The main kinds of cloud computing portability to consider are *data portability*, *application portability*, and *platform portability*. These are the portability respectively of data, application, and platform components.

Application interoperability between SaaS services and applications, and *platform interoperability* between PaaS services and platforms are important kinds of cloud computing interoperability to consider.

Applications can include programs concerned with the deployment, configuration, provisioning, and operation of cloud resources. Interoperability between these programs and the cloud resource environments is important. This is *management interoperability*.

Applications can also include programs such as app stores (for applications), data markets (for, e.g., openly available data) and cloud catalogues (e.g., reserved capacity exchanges, cloud service catalogs) from which users can acquire software products, data and cloud services, and to which developers can publish applications, data, and cloud services. In this Guide, all such

programs are referred to as *marketplaces*. *Publication and acquisition* of products is performed by platforms, including PaaS services, that interface to the marketplaces. This is the final important cloud interoperability interface.

The cloud computing portability and interoperability categories to consider are thus:

- Data Portability
- Application Portability
- Platform Portability
- Application Interoperability
- Platform Interoperability
- Management Interoperability
- Publication and Acquisition Interoperability

4.2 Data Portability

Data portability enables re-use of data components across different applications.

Suppose that an enterprise uses a SaaS product for Customer Relations Management (CRM), for example, and the commercial terms for use of that product become unattractive compared with other SaaS products or with use of an in-house CRM solution. The customer data held by the SaaS product may be crucial to the enterprise's operation. How easy will it be to move that data to another CRM solution?

In many cases, it will be very difficult. The structure of the data is often designed to fit a particular form of application processing, and a significant transformation is needed to produce data that can be handled by a different product.

This is no different from the difficulty of moving data between different products in a traditional environment. But, in a traditional environment, the customer is more often able to do nothing; to stay with an old version of a product, for example, rather than upgrading to a newer, more expensive one. With SaaS, the vendor can more easily force the customer to pay more or lose the service altogether.

Cloud introduces no new technical problems, but its different commercial arrangements can make the old technical problems much more serious.

4.3 Application Portability

Application portability enables the re-use of application components across cloud PaaS services and traditional computing platforms.

Suppose that an enterprise has an application built on a particular cloud PaaS service and, for cost, performance, or other reasons, wishes to move it to another PaaS service or to in-house systems. How easy will this be?

If the application uses features that are specific to the platform, or if the platform interface is non-standard, then it will not be easy.

Application portability requires a standard interface exposed by the supporting platform. As discussed under Application Interoperability (Section 4.5), this must enable the application to use the service discovery and information communication protocols implemented by the platform, as well as providing access to the platform capabilities that support the application directly. On a cloud PaaS platform, or a platform running on a cloud IaaS service, it may also enable applications to manage the underlying resources.

A particular application portability issue that arises with cloud computing is portability between development and operational environments. Cloud PaaS is particularly attractive for development environments from a financial perspective, because it avoids the need for investment in expensive systems that will be unused once the development is complete. But, where a different environment is to be used at run time – either on in-house systems or on different cloud services – it is essential that the applications can be moved unchanged between the two environments. Cloud computing is bringing development and operations closer together, and indeed increasingly leading to the two being integrated as *devops*. This can only work if the same environment is used for development and operation, or if there is application portability between development and operation environments.

4.4 Platform Portability

There are two kinds of platform portability:

- Re-use of platform components across cloud IaaS services and non-cloud infrastructure – *platform source portability*
- Re-use of bundles containing applications and data with their supporting platforms – *machine image portability*

The UNIX operating system provides an example of platform source portability. It is mostly written in the C programming language, and can be implemented on different hardware by re-compiling it and re-writing a few small hardware-dependent sections that are not coded in C. Some other operating systems can be ported in a similar way. This is the traditional approach to platform portability. It enables applications portability because applications that use the standard operating system interface can similarly be re-compiled and run on systems that have different hardware. It is illustrated in Platform Source Portability (Figure 3).

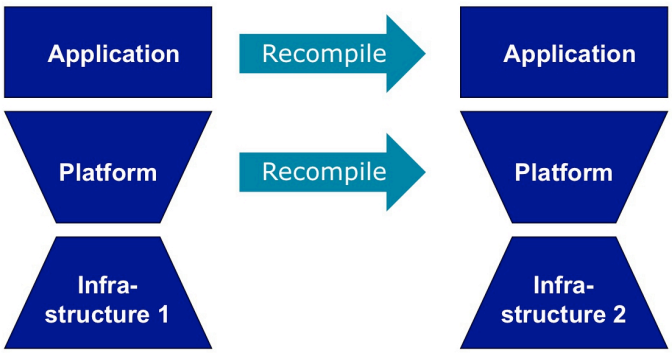

Figure 3: Platform Source Portability

Machine image portability gives enterprises and application vendors a new way of achieving applications portability, by bundling the application with its platform and porting the resulting bundle, as illustrated in Machine Image Portability (Figure 4). It requires a standard program representation that can be deployed in different IaaS use environments.

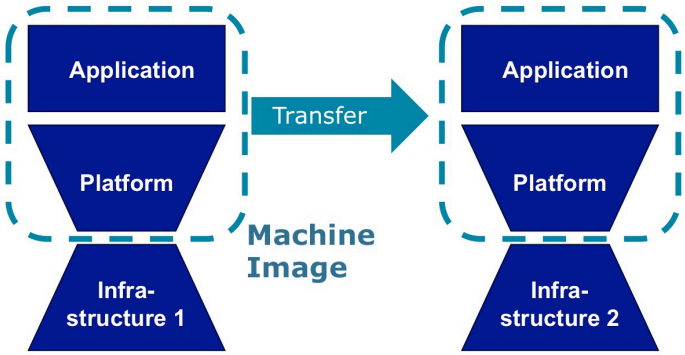

Figure 4: Machine Image Portability

4.5 Application Interoperability

Application interoperability is interoperability between application components deployed as SaaS, as applications using PaaS, as applications on platforms using IaaS, in a traditional enterprise IT environment, or on client devices. An application component may be a complete monolithic application, or a part of a distributed application.

Interoperability is required, not just between different components, but between identical components running in different clouds. For example, in a hybrid cloud solution, an application component may be deployed in a private cloud, with provision for a copy to be run in a public cloud to handle traffic peaks. The two components must work together.

Data synchronization is a particular issue when components in different clouds or internal resources work together, whether or not they are identical. Such components often keep copies of the same data, and these copies must be maintained in a consistent state. Communication between clouds typically has a high latency, which makes synchronization difficult. Also, the

two clouds may have different access control regimes, complicating the task of moving data between them. The design approach must address:

- Management of "system of record" sources
- Management of data at rest and data in transit across domains that may be under control of a cloud service consumer or provider
- Data visibility and transparency

Full interoperability includes dynamic discovery and composition: the ability to discover instances of application components, and combine them with other application component instances, at run time.

Cloud SaaS gives enterprises the possibility of acquiring new application capabilities quickly and easily, but much of the benefit of this is lost if costly integration work is needed to make the SaaS service interoperate with other applications and services that the enterprise uses.

Application components typically intercommunicate by invoking their respective platforms, which implement the necessary communications protocols. Protocol standards enable platform interoperability directly and are discussed under that heading. They are indirect enablers of application interoperability.

Application interoperability requires more than communications protocols. It requires that the interoperating applications share common process and data models. These are not appropriate subjects for generic standards, although there are specific standards for some particular applications and business areas.

There are, however, some design principles that improve application interoperability. Integration of applications designed following these principles still requires some effort, but is much less difficult and expensive than integration of applications that do not follow them.

4.6 Platform Interoperability

Platform interoperability is interoperability between platform components, which may be deployed as PaaS, as platforms on IaaS, in a traditional enterprise IT environment, or on client devices.

Platform interoperability requires standard protocols for service discovery and information exchange. As discussed above, these indirectly enable interoperability of the applications that use the platforms. Application interoperability cannot be achieved without platform interoperability.

Service discovery is currently used by a minority of applications, but is essential to achieve the highest levels of service integration maturity [OSIMM]. Standard service discovery protocols should be supported by platforms used by service registries and other applications.

Protocols for information exchange between platforms should support the establishment of sessions and transfer of session information, as well as information transport. (In the case of IaaS, the platform in question is not part of the infrastructure service but implemented on top of

it.) Session information might, for example, include the user's identity, the authorization level established by the user for access control purposes, the user's time-zone, the user's language, and the user's preferred cultural environment.

4.7 Management Interoperability

Management interoperability is interoperability between cloud services (SaaS, PaaS, or IaaS) and programs concerned with the implementation of on-demand self-service.

As cloud computing grows, enterprises will want to manage cloud services together with their in-house systems, using generic off-the-shelf systems management products. This can only be achieved if cloud services have standard interfaces.

These interoperability interfaces may provide the same functionality as the management interfaces mentioned under Application Portability (Section 4.3).

4.8 Publication and Acquisition Interoperability

Publication and acquisition interoperability is interoperability between platforms, including cloud PaaS services, and marketplaces (including app stores).

Cloud service providers often maintain *marketplaces* from which their cloud services can be obtained. Some also make associated components available. For example, an IaaS supplier may make available machine images that run on its infrastructure services. Some large user organizations, including governments, maintain app stores to which approved suppliers can publish programs, which can then be downloaded by the organization's departments. Some mobile device suppliers maintain app stores from which users can obtain apps to run on their devices.

Standard interfaces to these stores would lower the cost of cloud computing for software providers and users.

5 Distributed Computing Reference Model

This chapter describes the Distributed Computing Reference Model (DCRM). It contains an overview, descriptions of the components of the model, and sections on performance and security. It identifies the interfaces between the components. Portability and Interoperability Interfaces (Chapter 6) describes them in more detail.

5.1 Overview of the Model

The technical components and interfaces of the DCRM are shown in Distributed Computing Reference Model (Figure 5).

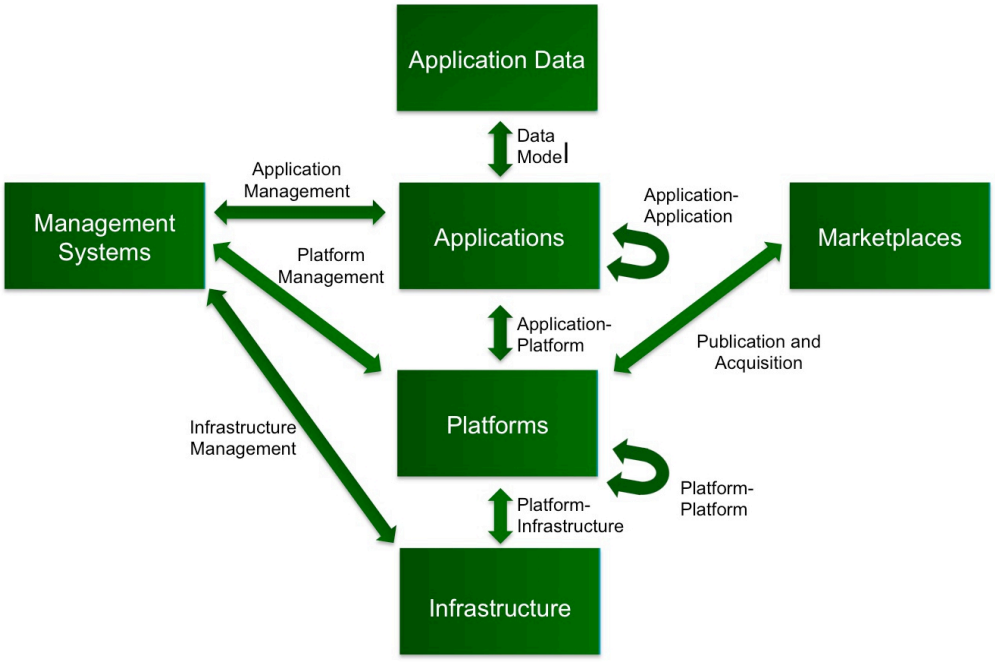

Figure 5: Distributed Computing Reference Model

The Management Systems and Marketplaces are particular kinds of application, shown separately because of their particular relationships to platforms and infrastructure.

The Application Data/Applications/Platforms/Infrastructure stack is present in enterprise systems, cloud systems, and user devices.

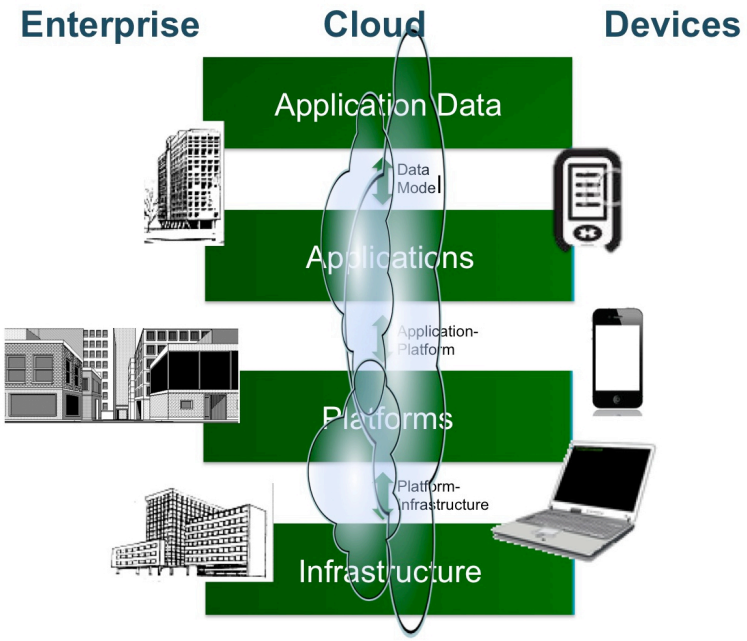

Figure 6: Enterprise, Cloud, and Devices

In cloud systems:

- Applications may be exposed as Software as a Service (SaaS).
- Platforms may be exposed as Platform as a Service (PaaS).
- Infrastructure may be exposed as Infrastructure as a Service (IaaS).

Different human actors use different components, as shown in Human Actors (Figure 7).

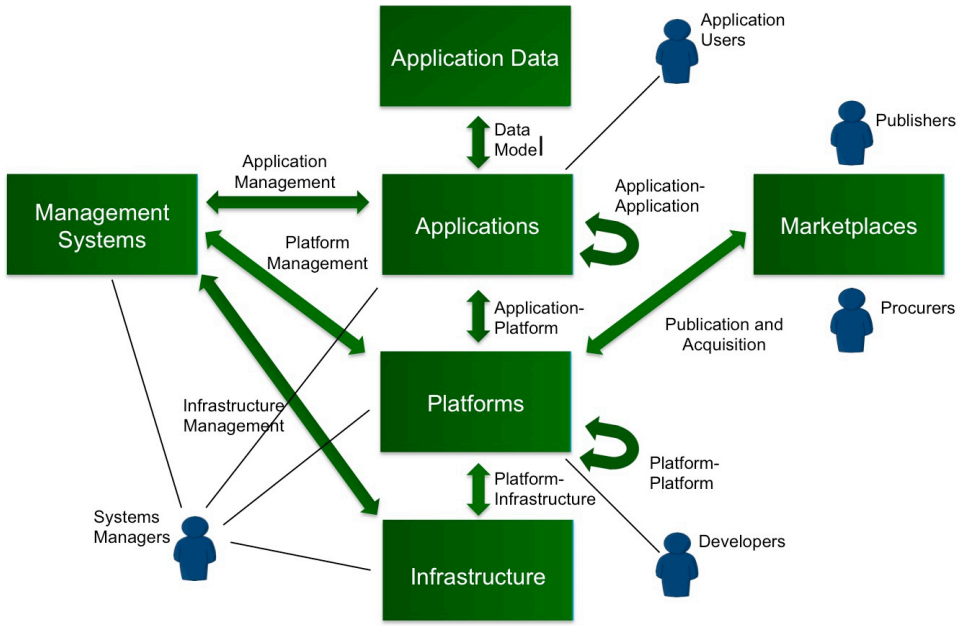

Figure 7: Human Actors

The following sections describe the model's components.

5.2 Application Data

An application may contain data that is regarded as a component in its own right. This is particularly the case for applications such as Relational Database Management Systems (RDBMS) that serve as data repositories for use by other applications.

The application interprets the data using a data model that defines the structure of the data and may include metadata that the application can use to interpret elements of the data.

Cloud computing is changing the ways in which enterprises store and use information.

Traditionally, data was created and stored by an enterprise for its own use. RDBMS provided the accepted best practice storage paradigm. Selected data, representing the final results of operations, was exposed externally. For example, a logistics company might send a customer a delivery record and payment invoice.

It is becoming increasingly common for companies to expose internal data. For example, logistics companies now often provide tracking information so that customers can see where there consignments are on the way to delivery.

Cloud computing raises the further possibility of collaborative use of data by an ecosystem of organizations. The ownership of such data becomes a matter of secondary importance. Accessibility of the information is the primary consideration.

Cloud computing also facilitates the assembly of really large collections of information ("big data"). Such a collection might come from a vast number of Internet transactions, or from sensors collecting real-time information about the environment (e.g., in weather stations) and connected to the Internet – the "Internet of Things". This information is often collected from, and stored in, a large number of locations, perhaps distributed globally.

5.3 Applications

An *application* is a deployed and operational IT system that supports business functions and services; for example, a payroll. Applications use data and are supported by multiple technology components but are distinct from the technology components that support the application. (See the Definitions section of the TOGAF Standard [TOGAF].)

In the DCRM, an application may be:

- A cloud SaaS service
- An enterprise application service
- A composition of cloud and enterprise application services
- An application program running on a server
- An application program running on a personal computer or mobile device (applications running on mobile devices are generally referred to as "apps")
- A web page or set of pages loaded onto a personal computer or mobile device (web pages are downloaded from servers and can include Javascript [JAVASCRIPT] to perform complex processing, as well as HTML [HTML] for information display)

In the SOA style, applications consist of collections of services. These may be explicitly programmed as service compositions; for instance, using the OASIS standard Web Services Business Process Execution Language (WS-BPEL). In this case, the WS-BPEL program is an application component, and the supporting process manager that enables this program to be created and executed is a part of the platform.

The role of applications in the DCRM is shown in Applications (Figure 8). (The interface to management systems is omitted from this figure for convenience.)

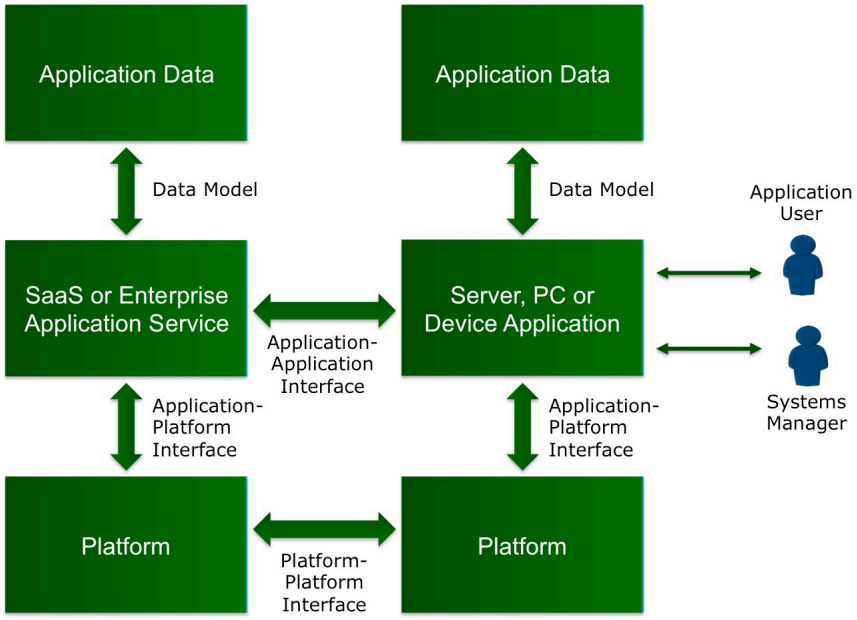

Figure 8: Applications

Online Shopping Application Example (Figure 9) illustrates how this could apply to a specific use-case: online shopping.

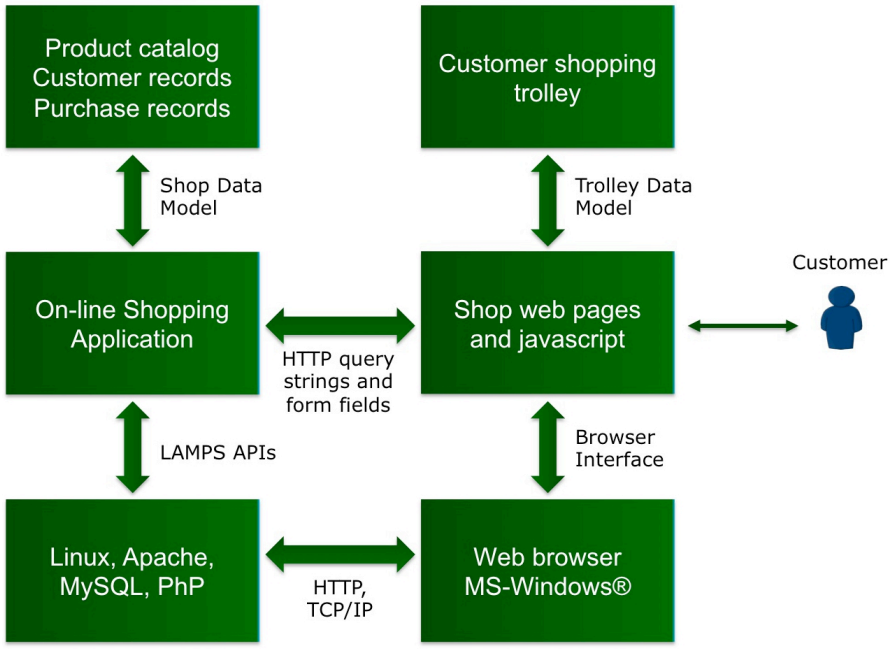

Figure 9: Online Shopping Application Example

Applications interface to:

- Application data through data models
- Each other through application-application interfaces
- Supporting platforms through application-platform interfaces
- Management systems through application management interfaces

SaaS services and enterprise application services generally interface to application users and systems managers via web pages displayed on PCs and mobile devices. Applications running on servers may interface to application users and systems managers in this way, but often have direct user interfaces.

5.4 Platforms

An application platform is the collection of technology components of hardware and software that provide the services used to support applications. (Again, see the Definitions section of the TOGAF Standard [TOGAF]. The term "service" is used in this definition in a general sense, not in the more specific sense of a software service.) It exposes an application platform interface to applications.

At the time that The Open Group was formed, the problem of application portability related to the re-use of application components across operating system platforms. The two consortia that combined to form The Open Group had made major contributions to the solution of this problem through their work on the UNIX operating system, which was then established as the *de jure* standard platform for application portability.

At that time, applications were typically large, monolithic software packages. This led to the problem of "information silos", which could be addressed by the replacement of monolithic applications by groups of loosely-coupled services in a Service-Oriented Architecture (SOA).

Also, an increasing amount of application processing is now performed on client devices. Web pages with Javascript are commonplace, as are mobile device "apps".

For these reasons, the problem of application portability is very different now to what it was 20 years ago.

5.4.1 Platforms in the DCRM

In the DCRM, a platform may be a PaaS service or an operating system platform on a server, PC, or mobile device.

Platforms interface to:

- Applications through application-platform interfaces
- Each other through platform-platform interfaces
- Infrastructure through platform-infrastructure interfaces

- Management systems through platform management interfaces
- Marketplaces through publication and acquisition interfaces

Platforms are used by developers to develop applications and other programs, and by systems managers to run and configure applications, either through management systems, or directly.

5.4.2 Platform Capabilities

Today, an application platform addressing portability must include more than a server operating system. A modern cloud portability platform could provide the following capabilities:

- Data storage and retrieval, including data management and search
- Data synchronization
- Process scheduling, including management of tasks and threads
- Inter-process communication, including session as well as transport capabilities
- Human-computer interface, including via web pages
- Identity management
- Access control
- Application access rights management to control application-platform service access
- Service discovery
- Service subscription
- Service tenancy allocation and binding
- Service composition
- Component and service templates and certification
- Transactionality
- Support for client-side processing
- Configuration and management of cloud resources, including load balancing and multiplexing, and policy management and scaling
- Software development
- Security configuration and management

In the future, application portability platforms may provide other capabilities, and in particular:

- Semantic translation

User identity management and access control are often particularly difficult when integrating services. Different services define different user roles, and have different ways of identifying

and authorizing users. They may hold similar information about users, represented in different ways. For a good user experience, these differences must be reconciled to enable single sign-on and automatic synchronization of user information across the services. This can be hard to do.

5.4.3 Platform as a Service (PaaS)

A cloud PaaS service exposes an application platform as a cloud service.

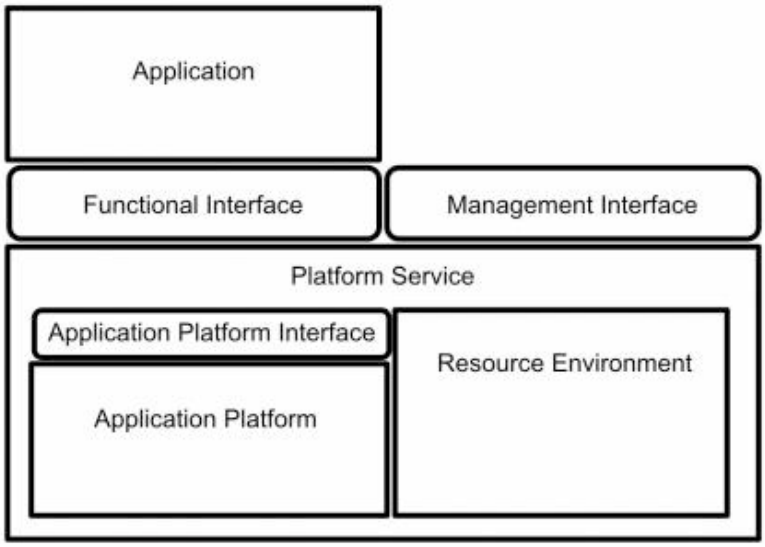

Figure 10: Platform Service

The application platform's application platform interface is exposed as the service's functional interface. It is a programmatic interface and supports applications running on the cloud platform service. The service environment also exposes a management interface. This is a web services interface that enables the cloud platform service to be configured and managed.

5.4.4 Web Service Interfaces

Cloud services are typically exposed as web services, with the platform handling the web service interface protocols. There are two different styles: *WS-I* and *raw HTTP*.

There is a set of standards for web service interfaces – WS-I (see [WS-I]) – but it is not universally used, and its use currently appears to be shrinking rather than growing. These standards are based on the use of XML message contents [XML] in a SOAP envelope [SOAP]. They include means of assuring the integrity, confidentiality, and non-repudiation of messages, and of conveying information that can be used for authentication and access control.

In practice, WS-I does not guarantee interoperability "out of the box" between products and services from different vendors. Experience shows that significant test and integration effort may be needed to make it work.

The approach that is increasingly displacing WS-I is the use of HTTP [HTTP] (or HTTP over a secure transport protocol – HTTPS) without SOAP as an upper layer, and with JSON [JSON] and other data formats in place of XML. This approach relies on HTTPS for assurance of

integrity and confidentiality. It makes no provision for non-repudiation, authentication, or access control, although the OAUTH authorization standard [OAUTH] can be used directly over HTTPS and is gaining some ground for authentication and access control.

The two styles are illustrated by the Example Use-Cases for WS-I and Raw HTTP (Appendix B).

5.5 Infrastructure

In the DCRM, infrastructure includes cloud IaaS services, hardware in servers, PCs and mobile devices, and virtualized hardware resources in enterprise systems.

A cloud infrastructure service makes hardware components available as cloud resources, generally through a virtualization layer.

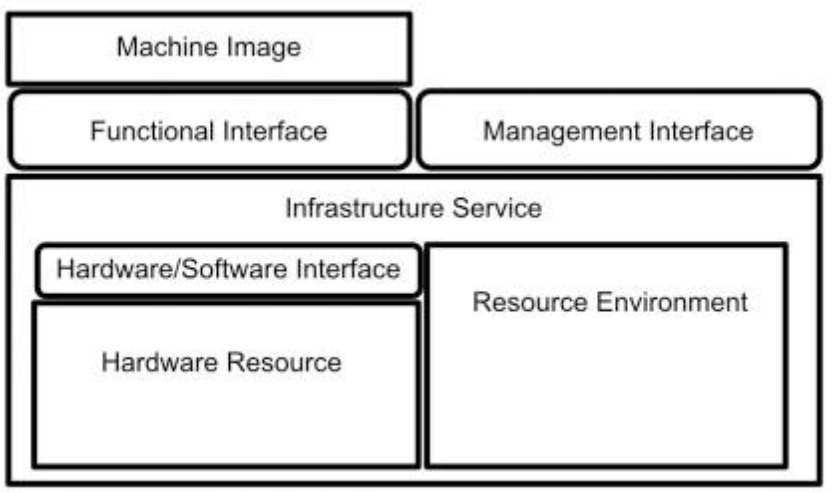

Figure 11: Infrastructure Service

The functional interface in this case supports the loading and execution of machine images. There is also a management interface that enables the hardware resources to be provisioned and configured, and enables machine images to be deployed on them.

Infrastructure interfaces to:

- Platforms through platform-infrastructure interfaces
- Management systems through infrastructure management interfaces

5.6 Management Systems

An enterprise may use a management system to manage its cloud and enterprise IT resources. It is convenient to be able to manage multiple resources – ideally all resources – with a single management system.

Management systems interface to:

- Applications through application management interfaces
- Platforms through platform management interfaces
- Infrastructure through infrastructure management interfaces

Management systems are used by systems managers to manage applications, platforms, and infrastructure.

5.7 Marketplaces

A cloud marketplace enables cloud providers to make their products available and enables cloud consumers to choose and obtain the products that meet their requirements. The products may be services, machine images, applications, or other cloud-related products. They can have associated descriptions, prices, terms of use, etc. so that consumers can select them and contract for their use.

Marketplaces interface to platforms through product publication and acquisition interfaces. These enable developers to put products into the market place, and systems managers to take them from the marketplace and put them onto their systems.

Marketplaces are used by publishers and procurers.

Publishers are people responsible for making products available to potential consumers. They post product descriptions, terms and conditions, and other information about the products. Where the products are available by purchase, rather than free-of-charge, this is a part of the sales function.

Procurers are people responsible for acquiring products for use by their enterprises, or by themselves. When acquiring a product, they typically create a contractual relationship between their enterprise (or themselves) and the product supplier.

5.8 Performance

Adequate performance is crucial to the usability of any system. Cloud computing introduces some particular performance issues. This section explains how they can be addressed in the context of the DCRM.

For an analysis of different types of workload, and their suitability for distributed processing in the context of cloud computing, refer to the discussion of *Workload Allocations* in the chapter on *Buying Cloud Services* in The Open Group Guide *Cloud Computing for Business* [CLOUD GUIDE].

System components that are “in the cloud” may physically reside in different places. The time taken to send a message between two cloud-based components, or between a cloud-based component and an in-house system, may be hundreds of milliseconds or seconds, rather than microseconds or milliseconds. The speed of data transfer is likely to be tens or hundreds of

kilobytes per second, rather than megabytes or gigabytes. End-to-end performance may be impacted by the concatenation of services across a distributed architecture that spans multiple clouds and in-house systems.

This can lead to two big problems:

- Unacceptably slow responses, due to the time taken for components to exchange many messages in order to action a request
- Inability to manage or move large data sets conveniently, due to the long transfer times

It might be possible to mitigate these problems to some extent, by positioning cloud services that communicate frequently with each other in the same data center of the same cloud supplier, but this runs counter to the principle that services should be from different suppliers and in different geographical areas for robustness and fault tolerance.

Developers of distributed computing applications are best advised to design them so that they are horizontally scalable, with components that can be replicated to meet increasing traffic, and that do not require exchange of messages that are frequent, large, or time-critical.

Where there is a requirement for application data to exist in more than one location, incremental changes should be copied as they occur, to avoid the need for bulk data transfers.

5.9 Security

The security of a system depends on the security of all of its components and on the security of all links between them, and security should be considered at all stages of system and component design. This section is not an exhaustive analysis of security in distributed systems; it just contains an overview discussion of some of the major aspects of the DCRM that relate to security. Designers must consult other material to gain an adequate understanding of current best practice. The Open Group Security Forum and Jericho Forum are one source of such material; see under *Security* in [OGBOOKS]. Another excellent source is the Cloud Security Alliance (CSA) [CSA], both for providing security assurance within cloud computing, and for use of cloud computing to help secure other forms of computing.

5.9.1 Tenant Separation

Infrastructure, platform, and application components may be *multi-tenanted*, meaning that different users, possibly from different enterprises, share resources. At the infrastructure level, multi-tenancy is generally achieved by virtualization; at the platform level it is generally achieved within the context of a multi-user operating system; and at the application level it is achieved by design of the application concerned. Generally, each tenant is unaware of the others, and uses the resources as though they were dedicated to him.

It is essential that secure separation is maintained, so that no tenant can obtain information about other tenants, the operation of their systems, or the content of their data, and so that no tenant can modify the desired operation of another's system.

Secure tenant separation is the responsibility of the operator of the multi-tenanted resource. Generally this is a cloud service provider.

5.9.2 Use of the Internet

In the DCRM, systems are accessed and information is exchanged between components over corporate intranets and the public Internet. It must be possible to do this in such a way that:

- Systems are only accessed by authorized principals.
- Information cannot be read or modified by a third party while in transit.
- Systems are not effectively disabled by Denial of Service (DOS) attacks.

All interfaces that require communication over the HTTP protocol are susceptible to normal web application security risks (for more information, see the OWASP Top Ten Project web security risks [OWASP]). However, the focus for our topic is the unique security vulnerabilities introduced by interoperability techniques.

- A web service with a published description in the WSDL [WSDL] offers rich information content to integration partners but provides users with a goldmine of information concerning the location of the web service, the methods it accepts (including parameters), and the format of the input. Malicious users can now fine-tune an attack based on the exposed WSDL, which can result in brute force password attack, SQL injection, session hijacking, and client-side bypassing of validations.
- Web services provide XML parsers [XML], which are vulnerable to XML injection. Users can input faulty information into an XML stream, which can override the correct values.
- Web service error handling can disclose information regarding application logic, which should normally be caught on another security layer.
- While SQL is mainly used to extract information from back-office databases, XPath [XPATH] is used by certain web services to query XML documents. As with SQL, XPath is susceptible to injection.

In the DCRM, it is the platform components that have the main capabilities that can be used to provide security against these threats.

Identity and access management require protocol exchanges between platform components, as a part of the platform-platform interface, and ability for applications to manage these through the application-platform interface. The WS-I style of platform-platform interface has better facilities for this than the raw HTTP style.

Security of information in transit requires secure protocols for their exchange. HTTPS is the most commonly used secure message exchange protocol. The raw HTTP style of platform-platform interface relies on it exclusively. The WS-I style includes additional secure messaging protocols (WS-Security).

Dealing with DOS attacks is inherently problematic and may require the participation of application components as well as platform components.

The WS-I and raw HTTP styles have very different security characteristics, as illustrated in Security Characteristics of WS-I and Raw HTTP (Table 1). WS-Security provides many solutions for web service security-related problems. However, the exposure of application logic

to the outside via web services introduces a new security risk, which cannot be mitigated by WS-Security.

Characteristic	WS-I Style	Raw HTTP Style
Support	Mainly supported by W3C and many large enterprises.	Mainly supported by developers.
Security Architecture	SOAP has an underlying security architecture, WS-Security.	REST has no security architecture, and is primarily relying on HTTP security standards.
Standard-specific Exploitations	Susceptible to all XML-related exploitations, such as manipulation/injection.	Susceptible to all HTTP/HTTPS exploitations, such as session ID hijacking.
Denial of Service (DOS)	SOAP is more susceptible to DOS attacks.	Firewalls can be employed to combat DOS attacks.
Man-in-the-Middle Attack	The security standard imposed by WS-Security provides a fairly good protection for man-in-the-middle attacks. Think of source authentication and message payload integrity. However, SOAP headers can be altered along the way, introducing new man-in-the-middle attacks (for example, references to non-existing locations).	Depends on the authentication methods used between point-to-point communications.

Table 1: Security Characteristics of WS-I and Raw HTTP

The WS-I style, using SOAP and WS-Security, is primarily focused on providing a security protocol for the data package itself (in this case, the encryption of a message). With the raw HTTP style, security is primarily focused on the transport of the data package, by having an encrypted connection between two points. If, for example, a new party C is introduced into a connection between two points A and B, SOAP can facilitate that the message arrives from A to B via C securely. The data package is at all times encrypted, and the authenticity of the package can be guaranteed. However, with the raw HTTP style, you would have a proper security line between A and C, and between C and B. In this case, C is able to view the package, change the contents of the package, or even delete it. This implies that the WS-I style should be used in case of a brokered architecture. Raw HTTP security is primarily relying on standard HTTP security options, such as TLS, SSL, cookies, and sessions.

5.9.3 Service Operation

The user of a cloud service, or other distributed computing service outside their control, must rely on the operator of that service not to read or modify their data or to modify the execution of their software. Ultimately, this is a question of ensuring that the service contract forbids such practices and trusting the supplier to honor it.

5.9.4 Resource Management

The infrastructure, platform, and application management interfaces must be secure, to prevent tampering with systems through their management interfaces.

5.9.5 Component Integrity

The components acquired from marketplaces must operate as specified, and must not include "Trojan Horses" or other features that could compromise a system's security.

Part of the value of a marketplace can lie in the security validation applied to products published in the marketplace, to protect the acquirers of those products.

6 Portability and Interoperability Interfaces

This chapter discusses the interfaces of the Distributed Computing Reference Model (DCRM), describing the standardization work that is needed for cloud computing portability and interoperability, and concludes with a summary.

6.1 Data Models

Data models describe the structure of data and are used by applications. They enable the applications to interpret and process the data. They apply to data that is held in store and accessed by applications, and also to data in messages passed between applications.

A data model may exist as:

- An undocumented shared understanding between programmers
- A human-readable description, possibly a standard
- A machine-readable description

While there are some standard data models, such as that defined by the ITU-T X.500 standards for directories, most data models are application-specific. There is, however, value in standardizing how these specific data models are described. This is important for data portability and for interoperability between applications and software services.

Relational database schema are the most commonly-encountered data models. They are based on the relational database table paradigm. A schema typically exists in human-readable and machine-readable form. The machine-readable form is used by the Database Management System (DBMS) that is the application that directly accesses the data. Applications that use the DBMS to access the data indirectly do not often use the machine-readable form of the schema; they work because their programmers read the human-readable form. The Structured Query Language (SQL) standard [SQL] applies to relational databases.

XML [XML] schemas and Document Type Definitions (DTDs) form another kind of data model. This is based on a different paradigm, of nested data components. Again, they exist in human-readable and machine-readable forms, with the machine-readable forms used by software that directly accesses the data, and software that accesses the data working because their programmers read the human-readable forms.

The semantic web standards can be used to define data and data models in machine-readable form. They are based on yet another paradigm, in which data and metadata exists as subject-verb-object triples. They include the Resource Description Framework (RDF) [RDF] and the Web Ontology Language (OWL) [OWL]. With these standards, all applications use the machine-readable form, and there is less reliance on understanding of the human-readable form by programmers.

Another, very simple, data model paradigm is emerging in web services and cloud computing. In this paradigm, data exists as a set of name-value pairs. This paradigm is assumed by many web service messages, and is the basis of some cloud SaaS "NoSQL" data stores.

The Universal Data Element Framework (UDEF) [UDEF] is a standard index that can be used in conjunction with data models of different kinds and facilitates integrated processing of the data.

6.2 Application-Application Interfaces

These are interfaces between applications. Increasingly, applications are web-based and intercommunicate using web service APIs. Other means of communication, such as message queuing or shared data, are and will continue to be widely used, for performance and other reasons, particularly between distributed components of a single application. However, APIs to loosely-coupled services form the best approach for interoperability. Other approaches are not considered in this Guide.

Some cloud service providers make client libraries available to make it easier to write programs that use their APIs. These client libraries may be available in one or more of the currently popular programming languages.

A service provider may go further, and make available complete applications that run on client devices and use its service ("client apps"). This is a growing phenomenon for mobile client devices such as tablets and smartphones. For example, many airlines supply "apps" that passengers can use to manage their bookings and check in to flights.

If a service provider publishes and guarantees to support an API then it can be regarded as an interoperability interface. Use of a library or client app can enable the service provider to change the underlying HTTP or SOAP interface without disrupting use of the service. In such a case a stable library interface may still enable interoperability. A service that is available only through client apps is unlikely to be interoperable.

A web service API is a protocol interface with four layers.

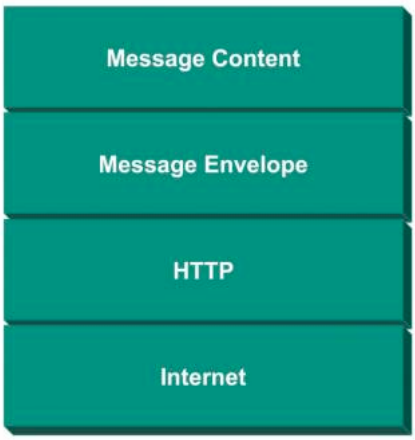

Figure 12: Web Service APIs

Applications are concerned with the highest layer, which is the message content layer. This provides transfer of information about the state of the client and the state of the service, including the values of data elements maintained by the service.

An application-application API specification defines the message content, the syntax in which it is expressed, and the envelopes in which it is transported.

The platforms supporting the applications handle the Internet, HTTP, and message envelope layers of a web service interface, and enable a service to send and receive arbitrary message contents. These layers are discussed under Platform-Platform Interfaces (Section 6.9) below.

Message content is application-specific, but:

- Some standardization may be appropriate within particular applications (for example, the Open Travel Alliance (OTA) [OTA] defines standard APIs to services such as hotel reservation, to enable collaboration between hotels and booking agents).
- The amount of application-specific processing required can be reduced by using standards for message syntax and semantics.

The W3C Extensible Markup Language (XML) [XML] is a generic syntax standard that is used in cloud service messages. Javascript Object Notation (JSON) [JSON] is an alternative that is increasingly gaining ground.

The Cloud Data Management Interface (CDMI) [CDMI] defined by the Storage Networking Industry Association (SNIA) is a standard application-specific interface for a generic data storage and retrieval application. (It is a direct HTTP interface, follows REST principles, and uses JSON to encode data elements.) It also provides some management capabilities.

Message contents are essentially data. Standards for describing data models can be applied to service functional interface message contents and improve service interoperability.

6.3 Application Management Interfaces

These are web service APIs, like Application-Application Interfaces (Section 6.2), but are presented by applications to expose management capabilities rather than functional capabilities.

Standardization of some message content is appropriate for these and other management interfaces. This is an active area of cloud standardization, and there are a number of emerging standards. Some are generic, while others are specific to applications, platform, or infrastructure management. None, however, appear to be widely adopted yet.

The SNIA CDMI [CDMI] is specific to data management applications. It can be used by administrative and management applications to manage data containers, accounts, security access, and monitoring/billing information.

There are two generic standards, TOSCA and OCCI, which apply to application management.

The OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) [TOSCA] is an XML standard language for descriptions of service-based applications and their

operation and management regimes. It can apply to complex services implemented on multiple interacting servers.

The Open Cloud Computing Interface (OCCI) [OCCI] of the Open Grid Forum is a standard interface for all kinds of cloud management tasks. OCCI was originally initiated to create a remote management API for IaaS model-based services, allowing for the development of interoperable tools for common tasks including deployment, autonomic scaling, and monitoring. The current release is suitable to serve many other models in addition to IaaS, including PaaS and SaaS.

6.4 Platform Management Interfaces

These are web service APIs, like Application-Application Interfaces (Section 6.2), that are presented by platforms to expose management capabilities.

The Cloud Application Management for Platforms (CAMP) [CAMP] is a PaaS management specification that is to be submitted to OASIS for development as an industry standard. It defines an API using REST and JSON for packaging and controlling PaaS workloads.

The generic application description standard TOSCA and the generic management standard OCCI apply to platform management also.

6.5 Infrastructure Management Interfaces

These are web service APIs, like Application-Application Interfaces (Section 6.2), that are presented by infrastructure services to expose management capabilities.

There are some standard frameworks that make it possible to write generic management systems that interoperate with vendor-specific products.

- The IETF Simple Network Management Protocol (SNMP) [SNMP] is the basis for such a framework. It is designed for Internet devices.
- The Common Management Information Service (CMIS) [CMIS] and the Common Management Information Protocol (CMIP) [CMIP] are the basis for another such framework. They are designed for telecommunication devices.
- The Distributed Management Task Force (DMTF) [DMTF] has defined a Common Information Model (CIM) [CIM] that provides a common definition of management information for systems of all kinds.

The DMTF is also defining the Cloud Infrastructure Management Interface (CIMI) model, with a REST interface and XML schema. It is working on a representation of this model using its generic management metamodel (CIMI-CIM).

The Virtualization Management (VMAN) standards [VMAN] of the DMTF extend DMTF standards for management of real resources to cover management of virtual resources. They include the Open Virtualization Format (OVF) [OVF], which is a standard for machine image

formats that may be loaded into IaaS services by management systems, and a key standard for platform portability.

The generic standards TOSCA and OCCI apply to infrastructure management also.

The Openstack Foundation has emerged as a major open source initiative to deliver software for building private and public clouds [OPENSTACK]. The formation of the Openstack Foundation in September 2012 gives the initiative a formal home, and it has significant support from major IT industry players. The OpenStack cloud operating system enables enterprises and service providers to offer on-demand computing, storage, and network resources, by provisioning and managing large networks of virtual machines. Resources are accessible via APIs for developers building cloud applications and via web interfaces for administrators and users. While Openstack is not a standards initiative, its administration APIs are likely to become *de facto* standards for cloud infrastructure management.

6.6 Publication Interfaces

These are interfaces between platforms and marketplaces that enable provider organizations to publish products and information about them into marketplaces.

Publication interfaces vary between marketplaces. A level of standardization is desirable for some aspects of product descriptions and especially for cloud service descriptions. These should cover three areas: functionality, quality of service, and conditions of contract.

The W3C standard for web service descriptions, the Web Service Description Language (WSDL) [WSDL], is a mature, proven standard. WSDL descriptions are, however, designed for interpretation by software programs rather than for being read by people. They are appropriate in an SOA that includes automated service discovery and selection, but need human-friendly presentation in a cloud computing marketplace where the choice is made by people.

Standard contractual clauses, and standard service-level descriptions, would be beneficial. There is some work in these areas.

The Open Data Center Alliance (ODCA) [ODCA] is developing a systematized approach to cloud procurement based on the identification of common usage models. This could result in the development or identification of standardized descriptions of quality-of-service parameters and conditions of contract.

The SLA@SOI Consortium [SLASOI] is developing a standard framework for Service-Level Agreements (SLAs), with a supporting toolkit.

The OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) [TOSCA] is intended to enhance the portability of cloud applications and the IT services that comprise them by enabling the interoperable description of application and infrastructure cloud services, the relationships between parts of the service, and the operational behavior of these services (e.g., deploy, patch, shutdown) independent of the supplier creating the service, and any particular cloud provider or hosting technology. It specifies a language for service template descriptions.

In addition to standards for the description of particular products, there is potentially a role for a marketplace catalog interface standard to define a common framework for product descriptions.

The concept of a marketplace as a system component is relatively new. It is too soon to expect mature standards for the publication and acquisition interfaces, but standards in this area will be beneficial, and their development should be encouraged.

6.7 Acquisition Interfaces

These are interfaces between platforms and marketplaces that enable consumer organizations to find out about products, and download them from marketplaces.

A download interface typically consists of an HTTP request that retrieves a file in the response. The format of the file is specific to the device platform and in some cases also to the component provider. There is, however, an emerging standard, the DMTF OVF, for machine image files [OVF].

Acquisition interfaces, other than for simple download, vary between marketplaces. As for publication interfaces, a level of standardization is desirable for some aspects of product descriptions and especially for cloud service descriptions, and standards for these could apply both to publication and acquisition interfaces.

6.8 Application Platform Interfaces

These are interfaces between applications and their supporting platforms. They are found in traditional solution stacks, in the resource environments of cloud and enterprise services, and in user devices. They are programmatic interfaces, for which standardization is needed to enable application portability.

As noted earlier, platforms today should do much more than traditional operating systems. The application-platform interface should enable applications to use the platform capabilities listed under Platforms (in Section 5.4).

A cloud application typically executes in a context that includes hardware, virtualization, operating system, programming language, and software libraries. The operating system provides an opaque layer that hides the hardware and (if present) virtualization from the application. Lower-level programming languages, such as C [C] expose the operating system to the application, but some applications use a higher-level language, such as Java [JAVA], that hides the operating system too. The software libraries are programming-language specific. For commonly required functions, it may be possible to obtain parallel libraries in the popular programming languages.

6.8.1 Operating Systems

Servers and workstations today mostly run operating systems that are versions of the UNIX operating system [UNIX], its derivative Linux [LINUX], or Microsoft Windows [MSWIN]. Web browsers on client devices provide an operating system for client applications; it does not have a name, but is quite well standardized across common browsers. Devices such as

cellphones and tablets have device operating systems; these are today mostly versions of Android [ANDROID] or iOS [IOS].

The UNIX operating system and Linux are quite similar, and there is good portability between them. Microsoft Windows is different, and applications written in low-level programming languages are not readily portable between it and UNIX systems or Linux. Web browsers and device operating systems are not directly compatible with the server/workstation operating systems, or with each other (although Android is to some extent based on Linux, and iOS is to some extent based on the UNIX operating system) but they do in some cases enable portability at the programming-language level.

6.8.2 Programming Languages

It is the programming language that is the key to applications portability today. Java provides good portability between UNIX systems, Linux, and Microsoft Windows, and some portability between these operating systems, web browsers, and Android. For web applications, PhP [PHP] provides portability between UNIX systems, Linux, and Microsoft Windows. Other languages such as Python [PYTHON] also provide portability between these operating systems, but are less widely used.

Web browsers support Java applets, rather than Java applications. Java applets are very similar to applications, and most code is directly re-usable between them. However, applets are not widely used. Programs to execute on web browsers are mostly written in Javascript [JAVASCRIPT] which, in spite of the name, is not at all the same as Java. A version of Javascript has been standardized as ECMAScript [ECMA262]. There is reasonable commonality of support for Javascript by major browsers, but this is not necessarily reflected in the official standard. The latest revision of HTML [HTML5] has recently gained significant traction in browser-based applications, both on the desktop and on mobile devices, and is now a *de facto* standard supported by all major browser vendors. This means that HTML/Javascript-based applications have a far more standardized platform than before.

Android supports a variant of Java that has some differences with the official version. The biggest difference is that special-purpose user interface libraries are supported in place of those supported by the official version.

iOS supports its own language – Objective-C – which is a higher-level language that is based on C but not compatible with Java.

6.8.3 Standard Libraries

There are two standard libraries that provide a significant part of the desired application-platform interface functionality: J2EE and .NET.

Java 2 Platform, Enterprise Edition (J2EE) is now officially known as Java Platform, Enterprise Edition [Java EE]. It is Oracle's enterprise Java computing platform. It includes Java classes that provide many aspects of server platform functionality, including:

- Data storage and retrieval, including data management and search
- Process scheduling, including management of tasks and threads

- Inter-process communication, including session as well as transport capabilities
- Platform-platform communication, either using SOAP or raw HTTP
- Human-computer interface, including via web pages
- Transactionality

The .NET Framework [.NET] is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It runs on servers and also includes versions for mobile or embedded device use. It has a common language infrastructure, whose purpose is to provide a language-neutral platform for application development and execution. In practice it is mainly associated with C# and other programming languages commonly used in Microsoft environments. It supports platform functionality broadly similar to that supported by J2EE.

6.8.4 Platform-Specific Libraries

Cloud PaaS platforms often have platform-specific software libraries that provide access to platform-specific functionality, and may even have platform-specific programming languages. Clearly, applications that use these libraries, or are written in these languages, will not be portable.

6.8.5 Resource Management

While the operating system provides an opaque layer that hides the hardware and (if present) virtualization from the application, there are some applications that take advantage of cloud on-demand self-service and rapid elasticity to configure and manage the underlying resources. Cloud platforms should give the ability to do this. The APIs of the Openstack open source cloud operating system [OPENSTACK] could become *de facto* standards for this.

6.8.6 Overall Standard

There are in fact a number of commercial and open source products that deliver various capabilities that should be provided by a modern distributed computing platform, but there is no overall service platform interface standard.

6.9 Platform-Platform Interfaces

These are protocol interfaces between platforms supporting applications. They include the lower three web services interface layers: *Internet*, *HTTP*, and *message envelope,* as explained under Application-Application Interfaces (Section 6.2). They also include interfaces for service discovery.

Standardization of these interfaces is needed to enable interoperability between platforms (which is a precondition for interoperability between applications).

The lowest web services interface layer is the Internet layer, at which information is exchanged using the Internet Protocol (IP) [IP], the Transmission Control Protocol (TCP) [TCP], and associated protocols from the Internet protocol suite. It provides basic information transport between Internet endpoints.

At the next layer is the HyperText Transfer Protocol (HTTP) [HTTP] of the World Wide Web, possibly used over a secure transport protocol (HTTPS). It provides information transport between web servers and their clients, with optional confidentiality and guaranteed integrity (by using secure transport).

Above this is the message envelope layer. This may be implemented using HTTP header fields, parameters and cookies, or using the message envelope parts of messages of SOAP [SOAP], layered on top of HTTP, as defined in the OASIS WS-I profile standards [WS-I]. Its functions include the maintenance of user sessions and user authentication. They may also (with SOAP) include additional security and identity management functions. Some services have parallel raw HTTP and SOAP APIs.

Users and managers can often interact directly with cloud services over the Internet or intranets using HTTP. The user browses and inputs to a set of web pages. Many cloud services have parallel human and programmatic interfaces; they have APIs that duplicate the function of their human-interface web pages. In these cases, the Internet and HTTP layers are the same for the human-computer interfaces and the APIs. Use of the raw HTTP style of interface may enable the envelope layer to be the same in these cases also.

The OASIS Universal Description Discovery and Integration (UDDI) standard [UDDI] applies to service discovery using web-based registries that expose information about a business or other entity and its technical interfaces. In conjunction with the WSDL described in Publication Interfaces (Section 6.6), it covers the technical interfaces to product functionality, but it is not a complete standard for the procurement interface and is not generally used by cloud marketplaces.

There is a proposal to standardize an XML language to describe HTTP-based web applications. This is the Web Application Description Language (WADL) [WADL]. It has been described as the REST equivalent of WSDL (which can also be used to describe REST web services).

6.10 Platform-Infrastructure Interfaces

Platform-infrastructure interfaces include hardware/software interfaces, used by low-level software components such as device drivers. It is desirable to allow for a wide variety of hardware devices, with different features, and there is no generic standard device interface. They also include interfaces to processors and memory provided by machine instruction sets. Again, a generic standard is not appropriate.

The OpenStack open source cloud infrastructure implementation [OPENSTACK] may provide a *de facto* standard for the interface to the cloud infrastructure. This would enable the portability of platform source code between cloud infrastructure implementations.

6.11 Summary

Summary of Interfaces (Table 2) lists the interfaces described in this section. It shows which components expose and use each interface, why standardization might be beneficial, and which standards or emerging standards might apply.

Interface	Exposed By	Used By	Type	Reason for Standardization	Standards
Data Model	Data	Applications	Description or Shared Understanding	Data Portability Application Interoperability	SQL XML RDF OWL UDEF
Application-Application Interfaces	Applications	Applications	Web Service API Content	Application Interoperability	N/A (but principles of arrangement are important)
Application Management Interfaces	Applications	Management Systems	Web Service API	Management Interoperability	CDMI TOSCA OCCI
Platform Management Interfaces	Platforms	Management Systems	Web Service API	Management Interoperability	CAMP TOSCA OCCI
Infrastructure Management Interfaces	Infrastructure	Management Systems	Web Service API	Management Interoperability Platform Portability (machine image)	CIMI-CIM VMAN OVF TOSCA OCCI Openstack
Publication Interfaces	Marketplaces	Platforms	Web Service API	Publication and Acquisition	ODCA SLA@SOI TOSCA
Acquisition Interfaces	Marketplaces	Platforms	Web Service API	Publication and Acquisition	HTTP OVF ODCA SLA@SOI TOSCA
Application-Platform Interfaces	Platforms	Applications	Programmatic	Application Portability	UNIX Linux MS Windows Android iOS Openstack
Platform-Platform Interfaces	Platforms	Platforms	Web Service API Envelope, HTTP and Internet Service Discovery	Platform Interoperability	TCP/IP HTTP SOAP WSDL WADL WS-I UDDI JSON

Interface	Exposed By	Used By	Type	Reason for Standardization	Standards
Platform-Infrastructure Interfaces	Infrastructure	Platforms	Various	Platform Portability (platform source)	Openstack

Table 2: Summary of Interfaces

7 Application Design Principles

Each application is different. Applications do not work with each other “out of the box”. Unless an application is stand-alone, effort is needed to integrate it into a system. The degree of interoperability of an application can be measured as its cost of integration.

A cloud application component typically comprises a set of operations with shared process, semantics, and access control. End users or other components invoke multiple operations in particular ways to obtain results. They construct information to send to the component, and interpret the responses, in a consistent way across all the operations. User roles are defined with different access rights to operations and information.

For example, a virtual meetings service might classify users as “attendee”, “moderator”, and “manager”. The sequence of operations to set up a meeting might be that a manager allocates a “meeting room” to a moderator, the moderator “prepares” a “meeting” in the meeting room, and each attendee “enters” the meeting room. The service has an API that includes “meeting room allocation”, “meeting preparation”, and “meeting attendance” operations, each of which is implemented as an HTTP request and response. The terms “attendee”, “enters”, “manager”, “meeting”, “meeting room”, “moderator”, “prepares”, etc. are used consistently in the product documentation, and in the API.

Such terms and procedures are, however, not consistent across applications. A different virtual meetings service might classify users as “participant” and “host”, with each host having a dedicated “conference bridge”. The sequence of operations might be that the host “starts” a “conference” and the participants then “join” it. The terminology and process is completely different.

Such an application is much more open and interoperable than a traditional silo application, and can be integrated with other products. For example, an enterprise could integrate both of the example virtual meetings services with a single calendar service to provide an enterprise virtual meeting scheduling capability. The effort required for such integration can be significant, because of the semantic and procedural variations.

It does not in general make sense to try to define standard functional interfaces for applications. Different virtual meeting service providers, for example, will have different procedures and terminology for their services, and these will be reflected in different interface definitions. There are, however, some application design principles that reduce the cost of integration, though this cannot be completely eliminated.

This chapter describes a number of application design principles that will reduce the cost of application integration for cloud computing.

7.1 Loose Coupling

Cloud application components should be loosely coupled with the application components that interact with them.

An application component should as far as possible be self-contained, with functions that are logically separate from those of other components. Interactions with other components should be simple, few, and not time-critical.

This final characteristic – that interactions are not time-critical – is particularly important for cloud computing, as communication with components that are "in the cloud" can be slow and their speed cannot be guaranteed.

7.2 Service-Orientation

Cloud applications should be service-oriented.

Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services [SOA].

A *service*:

- Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)
- Is self-contained
- May be composed of other services
- Is a "black box" to consumers of the service

A cloud application should be organized as a service, or a set of services, that may be a user of other services.

Low bandwidth, high-latency connections may mean that protocols commonly used in SOA are impractical, because of their high overheads. The principles of the service-oriented approach should nevertheless be followed whenever this is feasible.

7.3 Stable Interfaces

Cloud application components should have interfaces that do not change over time, or are such that any changes are backwards-compatible.

The cost of integration of a component should be considered over its lifetime, not just at its point of first use. A change to a component interface implies a need to re-integrate it with other components. The lifetime cost of a component whose interfaces change will be considerably more than its initial cost.

7.4 Described Interfaces

The interfaces to cloud application components should be clearly described.

The descriptions should be human-readable and may also be machine-readable.

Clear human-readable descriptions are essential for the acquisition of an application component and for its subsequent integration by developers.

Machine-readable descriptions enable dynamic discovery and composition.

7.5 Use of Marketplaces

Cloud products and services should be made available through marketplaces.

The marketplace (or app store) concept is becoming established in the world of cloud computing. A marketplace enables an enterprise or group of enterprises to ensure that the products and services that users need are available to them, and to exercise some control over the quality of those products and their suitability within the context of the enterprise or ecosystem architecture, while allowing users to choose between competing products.

The ability to restrict users to products and services that fit within an overall architecture will help to ensure portability and interoperability.

7.6 Representational State Transfer (REST)

Representational State Transfer (commonly known as REST) is an architectural style exemplified by the design of the World-Wide Web. It is described by Roy Fielding in Chapter 5 of his PhD Dissertation [FIELDING]. REST systems are characterized by having interactions that are client-server, stateless, and cacheable, by having uniform interfaces, and by being layered. They may also allow download and execution of code (e.g., as Javascript). The style is frequently referenced but widely misunderstood, with "REST API" often being equated to "API using raw HTTP".

While many REST APIs do use raw HTTP, the two concepts are by no means the same. In particular, use of raw HTTP does not guarantee that the service will not store user state information, as required by a truly "RESTful" approach.

For example, in a true REST system, each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. If this is followed strictly, when the user of an Internet shopping service requests to see his "shopping cart", the client must have remembered what is in the cart, and tell the server, rather than the server storing this information. Not all "RESTful" Internet shopping services actually work this way.

In fact, it is now normal practice for servers to store the identity of the individual using a client, and whether that client is logged in, as session information, rather than having the client authenticate itself for each interaction. Session information can also include such things as the

user's time-zone and preferred language. Not taking advantage of stored context on the server is interpreted as a guideline rather than an absolute prohibition.

The REST approach, though not necessarily applied with full rigor, is being increasingly used in cloud computing. It enables implementation of robust and scalable services. More than that, it is a simplifying principle that leads to the creation of loosely-coupled services with stable interfaces that are easy to describe.

7.7 BASE Transactions

Cloud applications should be designed as far as possible to perform transactions with the so-called *BASE properties*: Basically Available, Soft State, and Eventual Consistency.

Traditional transaction processing is based on the so-called *ACID properties*: Atomicity, Consistency, Isolation, and Durability. This approach means that transactions are reliable, but the required consistency cannot be obtained in distributed systems without sacrificing availability or robustness. (This proposition is known as the *CAP Theorem* [CAP].)

The BASE approach allows for replicated resources, so that at least one copy is always available, with different copies in slightly different states, but with synchronization between them so that there is eventual consistency as regards any particular piece of information.

Transactions in distributed systems that follow the BASE approach are not 100% reliable. For example, a purchaser may find, at check out, that an item that is "in the shopping cart" is unavailable because another user has purchased it meanwhile. This is a consequence that is generally accepted by today's Internet vendors and cloud service providers, in order to achieve robust, scalable systems.

There are some applications for which ACID transactionality is essential and, for those applications, it must be provided. For other applications, components providing parts of the transaction should be designed with BASE transactionality for interoperability with other components.

8 Conclusions and Recommendations

8.1 Conclusions

8.1.1 Importance of Cloud Portability and Interoperability

Cloud computing is a major development in information technology, comparable in importance with the mainframe, the minicomputer, the microprocessor, and the Internet. It has the potential to make an increasingly significant contribution to economic activity throughout the world. This potential will only be realized if cloud computing products and services are portable and interoperable.

8.1.2 Current State of Cloud Portability and Interoperability

The Internet is one of the foundations of cloud computing, and is an excellent example of how portability and interoperability can deliver massive benefits. But cloud computing is more than the Internet. The additional cloud components that store and process information, unlike the Internet and web components that they use to communicate and display it, are generally not portable and interoperable today.

8.1.3 Key Cloud Portability and Interoperability Categories

The key cloud computing portability and interoperability categories to consider are identified in the discussion of Cloud Portability and Interoperability (Chapter 4). They are:

- Data Portability
- Application Portability
- Platform Portability
- Application Interoperability
- Platform Interoperability
- Management Interoperability
- Publication and Acquisition Interoperability

8.1.4 The Distributed Computing Reference Model (DCRM)

The Distributed Computing Reference Model (Chapter 5) identifies the components of cloud-based systems that should be portable and interoperable. The interfaces between these components are discussed in Portability and Interoperability Interfaces (Chapter 6). Standardization of these interfaces is the first thing to consider as a means of achieving

portability and interoperability. It is appropriate for some but not all of the key categories. For one category, it is appropriate for the future but not yet.

Application portability, platform interoperability, and management interoperability can, as in traditional systems, be achieved by conformance of products to software interface and protocol standards.

8.1.5 Application Portability and Platform Interoperability

The platform capabilities required by modern cloud solutions are much greater than those of a traditional operating system, and the existing, well-supported operating system portability and interoperability standards are no longer sufficient. The web service protocol interface standards may be applicable, but are not necessarily appropriate. There are no standards for cloud software interfaces. A major effort is needed to define and popularize standards for cloud application portability and platform interoperability.

8.1.6 Management Interoperability

There are a number of initiatives to define cloud management interoperability standards. Work is needed to complete them, but what is most needed is to ensure that cloud products and services conform to them.

8.1.7 Data Portability and Application Interoperability

Data portability and application interoperability cannot generally be achieved by conformance to interface standards, because each application is different, and uses different data. Integration of applications, and porting of data between them, can, however, be more, or less, expensive, depending on how the applications are designed. Generic data model and semantic standards can also help reduce costs by enabling the use of powerful data management and semantic processing components.

There are some principles for the arrangement of application components whose adoption can improve data portability and application interoperability.

There are excellent data model standards, but new storage paradigms, requiring new standards, are emerging for cloud.

Improved understanding of semantic processing is needed for its benefits to be realized.

8.1.8 Marketplaces

Publication and acquisition of products and services through marketplaces is a new aspect of interoperability that is becoming increasingly important because of cloud computing. It is too early to define effective standards for them. It is important to develop and document good practices on which future standards can be based.

8.2 Recommendations

This section lists recommendations in the following areas:

- Platform-platform interfaces
- Application-platform interfaces
- Service descriptions
- Service management interfaces
- Data models
- Machine image formats
- Loose coupling
- Service-orientation
- Stable interfaces
- Described interfaces
- Marketplaces
- Representational State Transfer (REST)
- BASE transactions

These recommendations will:

- Help the industry to define standards and good practices to enable cloud portability and interoperability
- Help enterprises to maximize the portability and interoperability of the components in systems that use cloud computing

8.2.1 Platform-Platform Interfaces

Problem Summary

Platform-platform interfaces are the Internet, HTTP, and message envelope layers of web service APIs, which is the interface style of application-application interfaces, application management interfaces, platform management interfaces, infrastructure management interfaces, publication interfaces, and acquisition interfaces. A universally used standard for platform-platform interfaces would provide a major part of the basis for real cloud interoperability.

As described in Web Service Interfaces (Section 5.4.4), there are two styles of web service interface handled by platforms: WS-I and raw HTTP.

Each approach has strengths for specific applications. The choice between them also depends on factors such as the integration architecture and the capabilities of the implementation team.

Many small-scale integrations that originally used WS-I with SOAP and XML, because JSON was not mature enough at the time, are now moving towards raw HTTP and JSON because it is better suited to their needs. However, for enterprise-level integrations, SOAP is still king. This is illustrated in Example Use-Cases for WS-I and Raw HTTP (Appendix B).

Recommendations for Current Practice

CP1 Enterprises that are architecting sets of interoperating cloud services should use WS-I as the service platform interoperability interface between those services.

CP2 Enterprises that are developing cloud services that will be accessed by user devices should use direct HTTP with JSON as the service platform interoperability interface.

Recommendations for Standards Development

SD1 The industry should identify best practice in use of direct HTTP and JSON, including means of authentication and access control (such as OAUTH), and develop standard profiles for interoperability between service platforms using this approach.

8.2.2 Application-Platform Interfaces

Problem Summary

Application-platform interfaces are programmatic interfaces exposed by platforms. Their standardization would enable portability of applications across platforms. It could also enable applications to use WS-I and direct HTTP interchangeably as platform interoperability interfaces, thus simplifying the process of application development.

Currently:

- There are a number of programming languages that might be used for the interface.
- There is no agreement on what functionality is needed.
- There are no commonly-accepted application-platform interface standards that cover the full range of functionality; however, it might be agreed.

There are, however, products, both commercial and open source, that implement parts of the functionality, such as Enterprise Service Buses (ESBs), and some vendor-independent interface standards for part of the functionality, such as the Java Message Service (JMS).

Recommendations for Current Practice

CP3 Enterprises should seek to use cloud platforms with vendor-independent programming interfaces.

CP4 PaaS vendors stating that they support .NET or J2EE should say which versions they support.

Recommendations for Standards Development

SD2 A language-independent specification of a standard cloud application-platform interface should be defined. Instantiations of this should then be developed for the most widely-used programming languages.

8.2.3 Service Descriptions

Problem Summary

Human-readable service descriptions form part of publication interfaces and acquisition interfaces. Machine-readable service descriptions support dynamic service discovery and composition, and form part of platform-platform interfaces and application-platform interfaces.

There is an accepted standard for service descriptions, the Web Service Description Language (WSDL). This, however, has limitations.

- Its descriptions are machine-readable rather than human-friendly.
- It describes the functional characteristics of services, but does not cover non-functional characteristics such as quality of service and conditions of contract.
- It has no real ability to describe service data models.
- It applies to services that use the WS-I approach, but not to services that use the direct HTTP approach.

There are bodies working to develop standards for service descriptions that address some of these limitations. They include the companies that support the Web Application Description Language (WADL), the Open Data Center Alliance (ODCA), the SLA@SOI Consortium, and the OASIS TOSCA Technical Committee.

Recommendations for Current Practice

CP5 Enterprises developing services should produce clear human-readable descriptions of them, covering functional and non-functional characteristics.

CP6 Enterprises developing services using the WS-I approach should also produce WSDL descriptions of them.

CP7 Enterprises procuring services should insist on the availability of clear and stable human-readable descriptions and, for services using the WS-I approach, of WSDL descriptions.

Recommendations for Standards Development

SD3 The industry should work to establish best practice for human-readable service descriptions covering all service characteristics, building on the work of bodies currently active in this area.

SD4 The industry should work to establish standards for machine-readable service descriptions, including templates and component schemas. These standards should cover all service characteristics and parallel the human-readable descriptions. They should include or be linked to descriptions of service data models, and be applicable to services that use the direct HTTP approach as well as to those that use the WS-I approach. WSDL forms a good starting point for such standards.

8.2.4 Service Management Interfaces

Problem Summary

Service management interfaces enable management systems to manage cloud services. Standardization of these interfaces will enable the development of cloud management systems as commercial off-the-shelf products.

This is an active area of standardization. Initiatives include the DMTF Cloud Infrastructure Management Interface (CIMI) and Virtualization Management (VMAN) standards, the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA), the Open Grid Forum Open Cloud Computing Interface (OCCI), and the SNIA Cloud Data Management Interface (CDMI). The Openstack APIs may also provide *de facto* standards.

Recommendations for Current Practice

CP8 Enterprises developing cloud services should ensure that their management interfaces follow emerging standards where possible.

CP9 Enterprises procuring cloud services should look for services whose management interfaces follow emerging standards.

Recommendations for Standards Development

SD5 The industry should support the ongoing cloud management standardization work, including the work in the DMTF, OASIS, OGF, and SNIA, and the Openstack open source initiative.

8.2.5 Data Models

Problem Summary

Standardization of data models is important for data portability and for interoperability between applications and software services.

There are schema standards for particular storage paradigms, notably for RDBMS. These do not, however, cover the new "NoSQL" paradigms that are increasingly being used in cloud computing.

Also, the schema standards do not enable correspondences between different data models to be established, and this is crucial for interoperability. The semantic web standards and the UDEF can be used to define correspondence between data models, but their application is not widely understood, and they are little used.

Recommendations for Current Practice

CP10 Enterprises developing cloud services should describe their data models clearly, using text and applicable schema standards. The descriptions should be computer-readable and have good human-readable documentation. A well documented XML schema would achieve this, for example, but just using XML probably would not.

CP11 Enterprises procuring cloud services should look for clear data model descriptions.

Recommendations for Standards Development

SD6 The industry should establish best practice to describe correspondences between data models, should ensure that the standards in this area are fit for purpose, and should work to improve understanding of them.

8.2.6 Machine Image Formats

Problem Summary

Machine image formats are an important component of download interfaces and service management interfaces. The ability to load a machine image containing an application together with its application platform onto different cloud infrastructure services is a new form of portability that is made possible by cloud computing. A standard machine image format makes portability possible across different infrastructure service providers, as well as across infrastructure services of a single provider.

The DMTF OVF standard is designed to meet the need for a machine image format standard.

Recommendations for Current Practice

CP12 Enterprises developing cloud infrastructure services should evaluate the OVF standard and support it if feasible.

CP13 Enterprises developing cloud management systems should evaluate the OVF standard and support it if feasible.

CP14 Enterprises procuring cloud infrastructure services or cloud management systems should evaluate the OVF standard and look for support for it as appropriate.

Recommendations for Standards Development

SD7 The industry should work to ensure that the OVF standard is and remains fit for purpose, and to encourage its use.

8.2.7 Loose Coupling

Problem Summary

Tightly-coupled components are difficult and expensive to integrate, particularly over the lifetime of a system that undergoes change (as most do).

Recommendations for Current Practice

CP15 Cloud application components should be loosely coupled with the application components that interact with them.

8.2.8 Service-Orientation

Problem Summary

Interoperability requires that applications are easily integrated with each other, and that information can flow freely between them, to enable Boundaryless Information Flow within and between enterprises.

Service-orientation is a simple, business-focused principle that facilitates application integration and enables Boundaryless Information Flow. Cloud offerings are packaged as services (IaaS, PaaS, SaaS). Cloud platform-platform interfaces, whether in the WS-I or raw HTTP style, assume client-server interaction. Service-orientation encompasses and reinforces other principles – loose coupling, service descriptions, and described interfaces – that are recommended in this Guide.

Recommendations for Current Practice

CP16 Cloud applications should be service-oriented.

8.2.9 Stable Interfaces

Problem Summary

The cost of integration of a component, considered over its whole lifetime, will be high if the component interfaces change frequently.

Recommendations for Current Practice

CP17 Cloud application components should have interfaces that do not change over time, or are such that any changes are backwards-compatible.

8.2.10 Described Interfaces

Problem Summary

Components that do not have clear human-readable descriptions are hard to acquire and integrate. Components that do not have clear machine-readable descriptions are not suitable for dynamic discovery and composition.

Recommendations for Current Practice

CP18 The interfaces to cloud application components should be clearly described. The descriptions should be human-readable and should also be machine-readable if dynamic discovery and composition is intended.

8.2.11 Marketplaces

Problem Summary

Use of marketplaces and app stores is growing, but there are as yet no standards or established good practice for their operation. This means that product vendors must cater for the different requirements and practices of all the marketplaces in which their products appear, that customers must understand the different features of all the marketplaces that they use, and that marketplace operators are spending effort on unnecessary differentiation.

As this is a relatively new concept, standardization may be premature, but development of good practice should be encouraged.

Recommendations for Standards Development

SD8 Industry bodies should seek to identify the best practices for marketplace operation, with a view to defining standards and working with governments on any legislation that may be needed to underpin them.

8.2.12 Representational State Transfer (REST)

Problem Summary

There is a need for robust and scalable services that are loosely-coupled and have stable interfaces that are easy to describe.

Recommendations for Current Practice

CP19 Applications should be designed using the Representational State Transfer (REST) style, though without insisting on its full rigor.

8.2.13 BASE Transactions

Problem Summary

Traditional ACID transactionality cannot be achieved at the same time as availability and fault tolerance in a partitioned system. In distributed computing systems incorporating cloud-based components, partitioning, availability, and fault tolerance are of more importance than ACID transactionality in most cases.

Recommendations for Current Practice

CP20 Cloud applications should be designed as far as possible to perform transactions with the so-called BASE properties: Basically Available, Soft State, and Eventual Consistency.

A Cloud Computing and SOA

A.1 The SOA Reference Architecture

In the service-oriented style of Enterprise Architecture, monolithic applications are replaced by sets of loosely-coupled services. The SOA Reference Architecture (Figure 13) shows the conceptual building blocks of an SOA solution, and how they relate to each other. It can be used as a basis for specific solution models, and also for models of larger SOA systems, including those of enterprise SOA. In this context, the SOA Reference Architecture [SOARA] applies to the delivery of services from the cloud and can help to clarify the essential building blocks that can enhance interoperability and portability.

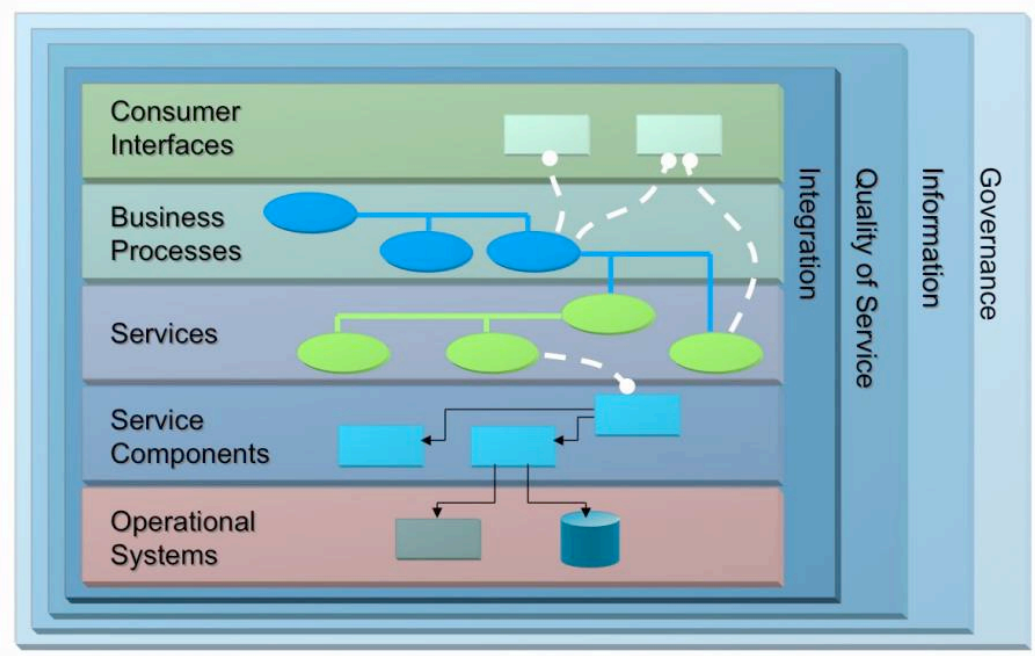

Figure 13: The SOA Reference Architecture

Five of the layers include processing that is specific to particular application functions. These, the functional layers, are shown on top of the other four, which constitute a generic services platform.

Existing application assets and other programs and resources are in the Operational Systems layer. This supports the Service Components layer, which contains software components that help to perform services and may leverage existing assets. Virtualization is performed in this layer. The central layer of the model is the Services layer, which contains the services, built from the components in the Service Components layer. The Business Processes layer contains compositions of the services in the Services layer. These are combinations of services that

support business processes, and can be formed dynamically. The final functional layer is the Consumer Interfacess layer, which contains the interfaces to people and software components that use the services.

The services platform comprises four layers. The Integration layer performs functions concerned with the integration of other building blocks. The Quality of Service layer handles quality aspects of system operation. This includes aspects relating to management and security. The Information layer is concerned with the processing of information within the system. The Governance layer is concerned with system functions that help ensure adherence to applicable regulations and policies. The Integration layer is directly concerned with interoperability. The other layers are important for interoperability and portability too.

A.2 Cloud Services

The essential concept of cloud computing, illustrated in Cloud Service (Figure 1), is that resources, which may be processing hardware, application platforms, or application software, are provided as a service, in a resource environment that enables them to be deployed and used, and with access via a communications network.

The environment has a *functional interface*, by which users can access the resource functions, and a *management interface*, by which users can deploy and configure them, and take advantage of the essential cloud computing characteristics of on-demand self-service, rapid elasticity, and measured service. This is shown in Cloud Service Interfaces (Figure 14).

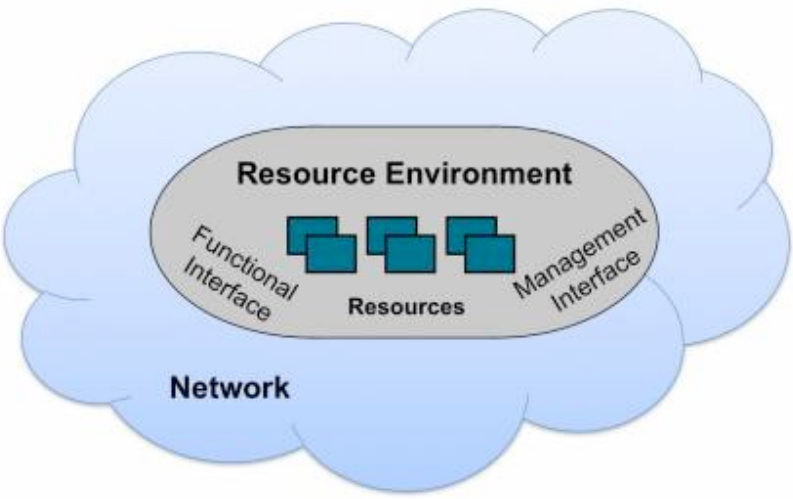

Figure 14: Cloud Service Interfaces

A.3 Application of the SOA Reference Architecture to Cloud Services

The resource environment can be organized in accordance with the SOA Reference Architecture, as shown in Resource Environment in SOA (Figure 15).

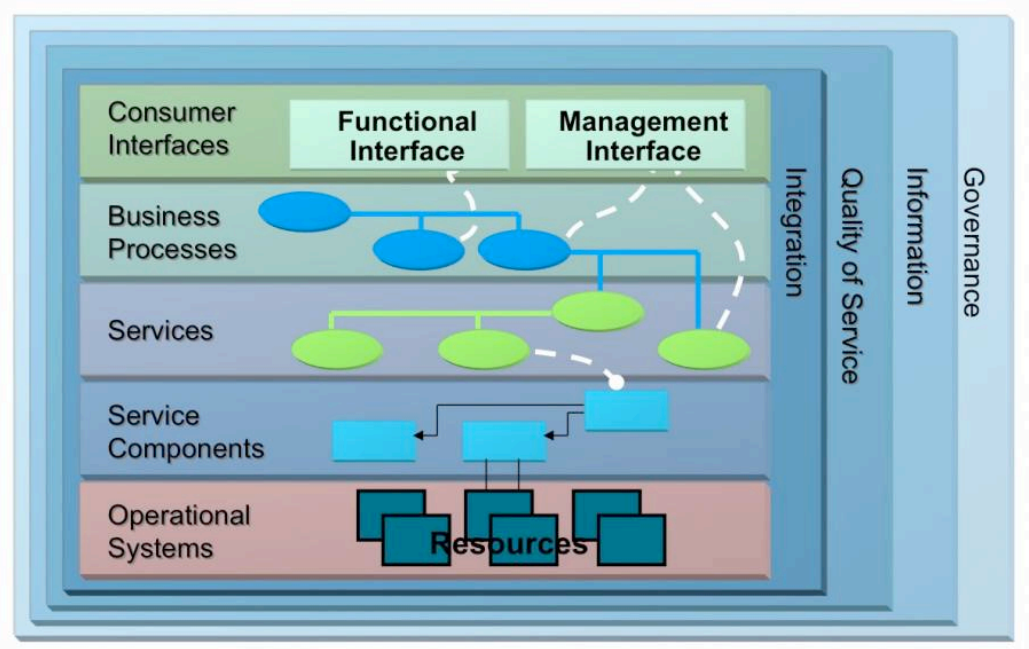

Figure 15: Resource Environment in SOA

The resources are in the Operational Systems layer, and the functional and management interfaces are in the Consumer Interfaces layer.

This applies to all forms of cloud computing: IaaS, PaaS, and SaaS. The relation to the DCRM, however, is different for SaaS and PaaS or IaaS.

For SaaS, the operational systems are applications programs. The functional layers of the SOA Reference Architecture are in the applications component of the DCRM, while the cross-cutting layers of the services platform are in the DCRM platforms component.

For PaaS, all layers of the SOA Reference Architecture are in the DCRM platforms component. For IaaS they are all in the infrastructure component.

B Example Use-Cases for WS-I and Raw HTTP

B.1 iPhone App

A custom iPhone app that reads your enterprise exchange account, to show you your agenda:

- Connection needs to be lightweight (mobile data charges).
- Data needs to be easily interpretable (mobile CPU is on average slower compared to desktops/laptops).
- Data and events are also on-demand (push from your exchange, or pull if you want to update your latest status).

JSON has the advantage over SOAP/XML in this case, as the JSON messages are significantly smaller, so less data is sent over the network. JSON is automatically parsed by JavaScript, so has significantly lower CPU usage and is easier to program in a browser environment.

Code Example: JSON

```
{"key":"someValue"}
```

Code Example: XML

```
<data><key>someValue</key></data>
```

B.2 Incident Management

You have an ITSM system and want it to communicate with your partners' similar ITSM system, so that you and your partner will both be able to see the relevant customer incidents and related CI information in the respective CMBD.

- Data needs to be reliable.
- Many business rules apply.

SOAP web services have the advantage here, because partners can access the CMDB web service and pull a lot of information; for example:

- Which fields the CMDB holds for a CI
- Which are required, which are optional
- What is the expected input per field (maximum length, integer, string, etc.)

All that is required to integrate a partner's system is to do a functional mapping of the fields (for example, to map the “CI identifier” fields of you and your partners' CMDBs).

Acronyms

The following acronyms are used in this Guide:

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Program Interface
BASE	Basically Available, Soft State, Eventual Consistency
BPaaS	Business Process as a Service
CAMP	Cloud Application Management for Platforms
CDMI	Cloud Data Management Interface
CI	Configuration Item
CIM	Common Information Model
CIMI	Cloud Infrastructure Management Interface
CMDB	Configuration Management Databases
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CRM	Customer Relations Management
CSA	Cloud Security Alliance
DBMS	Database Management System
DCRM	Distributed Computing Reference Model
DMTF	Distributed Management Task Force
DOS	Denial of Service
DTD	Document Type Definition
ESB	Enterprise Service Bus
FACE	Future Airborne Capability Environment
GDP	Gross Domestic Product
HTML	HyperText Markup Language

HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IP	Internet Protocol
ITSM	IT Service Management
J2EE	Java 2 Platform, Enterprise Edition
JMS	Java Message Service
JSON	Javascript Object Notation
NGO	Non-Governmental Organization
NIST	(US) National Institute of Standards and Technology
OAUTH	Open Authentication Protocol
OCCI	Open Cloud Computing Interface
ODCA	Open Data Center Alliance
OSIMM	The Open Group Service Integration Maturity Model
OTA	Open Travel Alliance
OVF	Open Virtualization Format
OWL	Web Ontology Language
PaaS	Platform as a Service
RDF	Resource Description Framework
REST	Representational State Transfer
SaaS	Software as a Service
SLA	Service-Level Agreement
SNIA	Storage Networking Industry Association
SNMP	Simple Network Management Protocol
SSL	Secure Sockets Layer
RDBMS	Relational Database Management System
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol

SQL	Structured Query Language
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOSCA	Topology and Orchestration Specification for Cloud Applications
UDDI	Universal Description Discovery and Integration
UDEF	Universal Data Element Framework
VMAN	Virtualization Management
WADL	Web Application Description Language
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Service Description Language
WS-I	Web Service Interoperability
XML	Extensible Markup Language