HCX: A Distributed OSGi Based Web Interaction System for Sharing Health Records in the Cloud

Sabah Mohammed, Daniel Servos and Jinan Fiaidhi
Department of Computer Science, Lakehead University, Thunder Bay, ON P7B 5E1, Canada
{sabah.mohammed, dservos, jfiaidhi}@lakeheadu.ca

Abstract

WITH the maturity of Web Services and Enterprise Service Oriented Architectures (SOA), new delivery and Web interaction models are now demonstrating how services can be traded outside traditional ownership and provisioning boundaries. The value of SOA comes from having an architecture that readily accommodates change. The more your business changes, the more SOA pays for itself. However, the initial build-out of SOA, prior to business change or service sharing, is cost-ineffective. By incorporating cloud computing in SOA, the time to value is shortened because you leverage 'other people's work' as well as saving on infrastructure cost by leveraging on demand cloud based infrastructure services. This article outlines a distributed Web interactive system for sharing health records on the cloud using distributed OSGi services and consumers, called HCX (Health Cloud eXchange). This system allows for different health record and related healthcare services to be dynamically discovered and interactively used by client programs running within a federated private cloud. A basic prototype is presented as proof of concept along with a description to the steps and processes involved in setting up the underlying infrastructure. Finally some future directions for using this infrastructure are illustrated.

1. Introduction

RECENT foray by lead vendors like Amazon, Google, IBM, Sun Microsystems, Microsoft etc. into cloud computing, advocate for a universal sharing scenario where enterprises are not required to own any infrastructure or applications of their own. This is enabled by powerful usage of SOA - namely infrastructure available as a set of usable pluggable services. Actually, cloud computing offers an ideal alternative to heavy investment in server and hardware infrastructure that most information systems require, as well as an easy means of scaling resources to meet changes on demand. In periods of economic uncertainty instututions look at the cloud computing option to enable them to continue to meet end-user and business demands for IT services. Cloud computing is an opportunity for business to implement low cost, low power and high efficiency systems to deliver scalable infrastructure. It increases capacity and expands computing capabilities without heavy investment in infrastructure, training or software licensing [1].

Triggered by an aging populace, rising population numbers, and an increased awareness of the need to create a continuum of care amongst the health community, healthcare institutions and users now desire quick and easy access to accurate patient data. Cost and poor usability have been cited as the biggest obstacles to adoption of health IT – especially Electronic Health Records (EHR) systems – and has resulted in problematically-low EHR adoption rates. Cloud Computing may help in eliminating this cost, and IT maintenance burdens that are often beyond the reach of small medical practices. Elimination of barriers to EHR adoption such as costly hardware infrastructure, while at the same time ensuring security and privacy of protected health information, are all positive results of using the "cloud" approach [2]. This article outlines the Health Cloud Exchange (HCX) system which provides a cloud based EHR systems for sharing health records between services and consumers that enable both clients and third party healthcare information systems (including other EHR systems) to share records in a dynamic way that automatically adapts to changes in the cloud using the Java based OSGi framework [3].

2. Identifying Suitable Cloud Technologies

MANY technologies have been developed to support cloud computing at each layer (Application, Platform, and Infrastructure) which are offered as services, software packages and frameworks. Major vendors, including top firms such as Amazon, Google, and Microsoft, jumped on the cloud bandwagon with a wide-range of offerings. With leading companies still joining the movement — including IBM, HP, and Sun Microsystems — cloud computing has moved from a cottage industry to one of the bigger growth areas in the computing business, just as the industry as a whole begins to take serious lumps from the recession [4]. Figure 1 illustrates the landscape of the cloud computing technologies.

978-0-7695-4191-4/10 $26.00 © 2010 IEEE
DOI 10.1109/WI-IAT.2010.26

Figure 1: Cloud Computing Technologies -- modified image from [4].

In our HCX prototype we used two major open source frameworks; Eucalyptus (http://www.eucalyptus.com/) (for providing private cloud infrastructure as a service) and Apache CXF DOSGi (http://cxf.apache.org/) (for building distributed OSGi based services). Eucalyptus is an open-source software platform that implements IaaS-style cloud computing using the existing Linux-based infrastructure found in the modern data center. Its interface is compatible with and based on Amazon's AWS interface making it possible to move workloads between AWS and a private Eucalyptus based cloud significant modification to the code that comprises them. Eucalyptus supports a variety of virtualization technologies including the VMware, Xen, and KVM hypervisors for the creation and management of virtual servers. Compared to other private cloud frameworks such as Nimbus (http://www.nimbusproject.org/) and abiCloud (http://abicloud.org), Eucalyptus was chosen as it has a stronger community support, larger amount of documentation and comes packaged in the easy to install Ubuntu Enterprise Cloud Linux distribution. Apache CXF DOSGi was chosen as it is the reference implementation for distributed OSGi and uses a ZooKeeper (http://hadoop.apache.org/zookeeper/) based cluster as a central service registry which enables simple and scalable service discovery while keeping the advantages of a distributed system (e.g. not rely on a single point of failure).

The hardware infrastructure consisted of several identical Xeon based servers that were assigned static IPs and made accessible to each other over the Lakehead University local area network to form a Eucalyptus based private cloud computing test bed. The Eucalyptus Cloud Controller, Walrus, and Storage Controller and Cluster Controller services were installed on one of the servers to provide scheduling, control, S3 based bucket storage, EBS based storage and a front end for the cloud that all cloud related queries would go through. On the remaining servers, the eucalyptus Node Controller service and Xen kernel were installed to provide virtualization services to the cloud. This set up is shown in Figure 2.

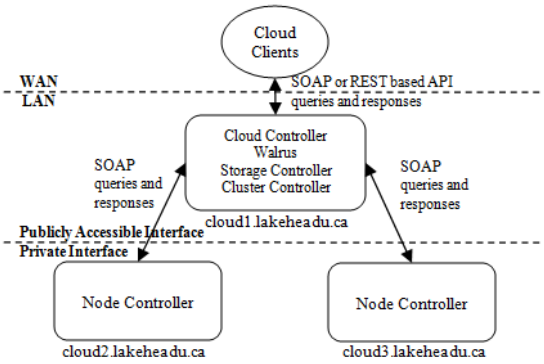

Figure 2: Lakehead University's Eucalyptus Private Cloud Set-up.

In addition to Eucalyptus and Apache CXF DOSGi, several supporting technologies and cloud management tools were required during the implementation of the prototype (e.g Xen hypervisor (www.xen.org/) to support virtualization of machine images for the cloud). BIND (www.isc.org/software/bind); an open source DNS implementation was used to dynamically map hostnames to virtual machines operating within the cloud, while Apache Hadoop and its Zookeeper subproject were installed on several machine images to support the use of Apache CXF DOSGi's discovery services.

3. Sharing EHRs over HCX

A large number of EHRs specifications and standards exist for storing, processing and transmitting patient healthcare records such as HL7 [5], DICOM [6], Continuity of Care Record (CCR) [7] and Continuity of Care Document (CCD) [8]. The HCX system described in this paper uses the CCR format and specification to encapsulate patient health care data sent between OSGi based modules. CCR is an XML document that is comprised of three main elements: header, body, and footer. The header contains Meta data about the document including the CCR format version, the date the document was created, a unique id identifying the document and a unique id identifying the patient. The body of the document contains detailed information about procedures, immunizations, family history, insurance information, medications, vital signs, test results, and medical alerts of and relating to the patient. The footer contains a list of actors (persons, organizations and information systems) referenced throughout the document by an actor id and adds additional information such as name, address, email,

etc. This actors section also provides an easy means to anonymize the document as removing the actors section will only leave references to actor ids rather than complete names or addresses of persons. Also contained in the footer section is a reference section that allows external documents such as DICOM images to be linked with a CCR based health care record, a comments selection which allows comments to be placed on different CCR elements, and a signatures section for digital singing of elements in a CCR document. CCR was chosen as its XML based nature makes it easy to parse and implement while still being powerful enough to contain most critical patient information and health history. Being XML based also allows CCR documents to be easily transformed into human readable documents using XSL Transformations. CCR also helps with interoperability, as XML is platform and language independent and can be implemented in any system with an XML parser available. The CCR standard is currently being used by multiple EHR systems including Google Health (https://www.google.com/health) and Microsoft's HealthVault (http://www.healthvault.com/), particularly for sharing data with other EHR based systems making interoperability much less of an issue.

The HCX system is based around the OSGi framework and consists of modular services (called CCRStores) dedicated to sharing CCR formatted health records with consumers in the same private cloud. Consumers query CCRStores for either a listing of available health records or a specific record using a shared interface implemented by all CCRStores. CCRStore consumers can be bridges to outside EHR systems or applications that interact directly with clients (e.g. web based client for viewing EHRs in a private cloud). CCRStores themselves can be databases of health records stored on the cloud, bridges to outside EHRs systems and databases, or EHR systems on the same private cloud. CCRStores register themselves as distributed OSGi services that can be stopped, started, installed or removed at any time, like any other OSGi service. Consumers of CCRStores use Distributed OSGi based discovery and tracking to find new CCRStores and keep an updated list of current stores available on the cloud. This allows for loose coupling between the consumers and stores as consumer need only know about the standard CCRStore interface and the location of the service registry to use any store that becomes available on the cloud, including bridges to other systems outside of the cloud. Figure 3 illustrates the general layout of the Apache CXF DOSGi framework on which HCX is built.

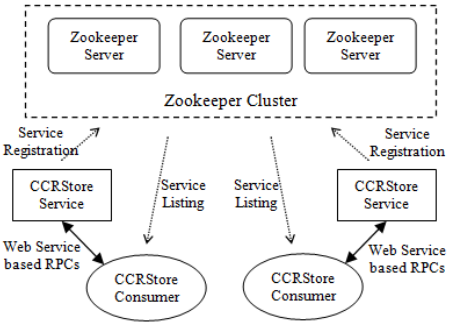

Figure 3: Layout of DOSGi based services, consumers and registry.

4. The HCX Architecture

DISTRIBUTED OSGi under Eucalyptus provides a good basis for building the HCX system as OSGi's modular service based infrastructure already accomplishes several goals of an effective distributed EHRs system (distributed, private, loose coupling and modularity), being cloud based provides the dynamic scalability needed to overcome costly infrastructure while meeting any demand. CCRStores are deployed on virtual servers within the cloud which may be dynamically created or destroyed as needed (ie. In the event of high demand multiple instances of a CCRStore repository maybe be created and connections load balanced between them). Each CCRStore implements the same standard interface (see Figure 4) and is also an OSGi service that registers it's self with a central registry once successfully started. Consumers use the same DOSGi framework and CCRStore interface to discover and consume the registered CCRStores.

CCRStores come in three main types; *repositories, bridges and adaptors*. Repositories are databases of health records stored on the cloud which are independent of any other EHR system and primarily accessed through HCX. Bridges provide a link to resources outside of the private cloud which may be other EHR systems or repositories. For example a bridge may be created that retrieves health records from Google Health based on some query from a consumer. Adaptors provide the middleware between an EHR system on the same private cloud and consumers, offering the records available in the EHR as a CCRStore. The difference between repositories, bridges and adaptors is transparent to consumers as all implement the same interface and are accessible in the same way.

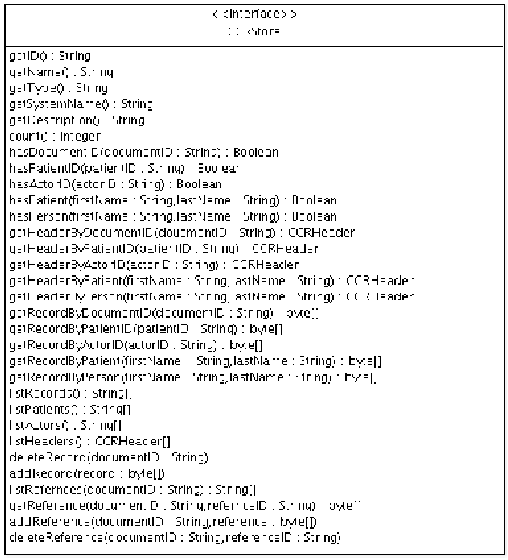

Figure 4: The CCRStores interface.

Consumers of CCRStores also come in three main types; *client bridges, system bridges and adaptors*. Client bridges consume CCRStore resources and provide them directly to clients in a user friendly interface. For example a consumer may provide a web based interface to clients that shows all available EHR discovered on the cloud. System bridges provide outside access to existing EHR systems that want to import records from the cloud. Adaptors allow EHR systems on the same cloud to consume the records available in CCRStores. A single EHR system may have both a consumer and CCRStore services so that it may both retrieve and provide records to and from the cloud.

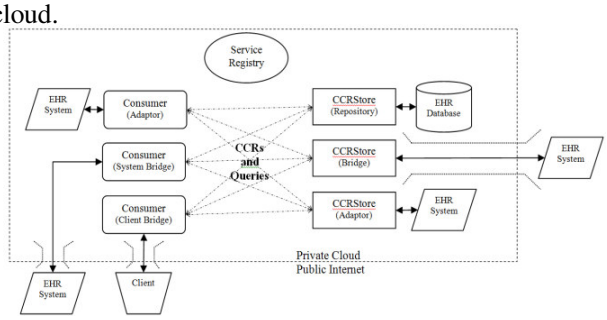

Figure 5: HCX Interaction System Diagram

Figure 5 depicts these interactions between the different HCX services, consumers and existing EHR systems. The service registry consists of one or more Zookeeper servers which are queried on the registration of a new service to create a file detailing information on how to connect to and use the service. Consumers may then discover services by querying the Zookeeper cluster for the files. Consumers send queries for data from the CCRStore and are responded to with one of the following; an exception indicating that the query or operation has failed, a list of available records, a CCR formatted record or a Java object representing the header of a CCR file. In addition to queries for data, consumers may also request the deletion or addition of records. All queries and operations performed on a CCRStore are atomic and only change the underlying records if the operation was successful. Figure 6 shows an example service registry for a CCRStore service.

```
#
#Mon Apr 12 13:07:46 EDT 2010
osgi.remote.endpoint.location=http\://192.168.105.250\:
51874/display
service.id=60
org.apache.cxf.ws.address=http\://192.168.105.250\:5187
4/display
objectClass=[Ljava.lang.String;@1623820
endpoint.id=http\://192.168.105.250\:51874/display
service.exported.interfaces=*
service.exported.configs=org.apache.cxf.ws
osgi.remote.endpoint.id=29334005-828d-459f-a98c-
6cf7d62a1e62

cZxid = 236
ctime = Mon Apr 12 13:07:59 EDT 2010
mZxid = 236
mtime = Mon Apr 12 13:07:59 EDT 2010
pZxid = 236
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 83299681533886489
dataLength = 405
numChildren = 0
```

Figure 6: osgi/service_registry/ccrstore/CCRStore/192.168.105.250#51874##display

Table 1 lists the primitive methods that must be supported by all CCRStore implementations:

getID Parameters: None Returns: String	Returns a globally unique ID for the CCRStore, this can either be based on the IP address and port number the CCRStore service is being offered on or some other globally unique ID accessible via the DOSGi implementation.
getName Parameters: None Returns: String	Returns a name that is unique to the CCRStore implementation but not necessarily unique among all currently running services
getType Parameters: None Returns: one of "repo", "bridge", or "adaptor"	Returns the type of the CCRStore. This may be one of "repo" (for local EHR repositories), "bridge" (for bridges to outside systems), or "adaptor" (for local EHR systems).
getSystemName Parameters: None Returns: String	Returns the name of the EHR system or repository this CCRStore is serving records for. For example if this CCRStore is a bridge to a set of Google Health records this may return "Google Health".
count Parameters: None Returns: int	Returns the number of records stored on this CCRStore that are accessible to the querying consumer.
hasDocumentID Parameters: Document ID (String) Returns: boolean	Checks if this CCRStore has a health record such that if the record was formatted as a CCR the document ID of that CCR document matches the given document ID. Document IDs should be globally unique and not contain any indentifying information.
hasPatientID Parameters: Patient ID (String) Returns: boolean	Checks if this CCRStore has a health record such that if the record was formatted as a CCR the actor ID of the patient would match the given patient ID. Actor IDs should be globally unique and not contain any identifying information.

hasActorID Parameters: Actor ID (String) Returns: boolean	Checks if this CCRStore has a health record such that if the record was formatted as a CCR there would be an actor ID in the actors section that matches the given actor id. This actor may be the patient them self, a physician somehow related to the patient's record or any other person or organization listed in the actors section of the CCR document.
hasPatient Parameters: First name (String), Last name (String) Returns: boolean	Checks if there is a health record available in the CCRStore for a patient matching the given first and last names. One of the parameters may be left as null to only search for a matching last or first name.
hasPerson Parameters: First name (String), Last name (String) Returns: boolean	Checks if the CCRStore has a record such that if it was formatted into a CCR document the actors section would contain a person element matching the given first and last name. One of the parameters may be left as null to only search for a matching last or first name.
getHeaderByDocumentID Parameters: Document ID (String) Returns: CCRHeader or null	Returns the header of the CCR that has a document ID matching the given ID. If no such CCR exists on this CCRStore, null is returned.
getHeaderByPatientID Parameters: Patient ID (String) Returns: CCRHeader or null	Returns the header of the CCR for a patient with an actor id matching the given patient id. If no such CCR exists on this CCRStore, null is returned.
getHeaderByActorID Parameters: Actor ID (String) Returns: CCRHeader[] or null	Returns the headers of any CCR document with at least one actor ID found in the actors section matching the given actor ID. If no such CCR exists on this CCRStore, null is returned.
getHeaderByPatient Parameters: First name (String), Last name (String) Returns: CCRHeader[] or null	Returns the headers of any CCR document for a patient with a matching first or last name. One of the parameters may be left as null to only search for a matching last or first name. If no such CCR exists on this CCRStore, null is returned.
getHeaderByPerson Parameters: First name (String), Last name (String) Returns: CCRHeader[] or null	Returns the headers of any CCR document where a person listed in the actor's selection matches both the given first and last name. One of the parameters may be left as null to only search for a matching last or first name. If no such CCR exists on this CCRStore, null is returned.
getRecordByDocumentID Parameters: Document ID (String) Returns: byte[] or null	Returns the CCR record as a byte array that has a document ID matching the given ID. If no such CCR exists on this CCRStore, null is returned.
getRecordByPatientID Parameters: Patient ID (String) Returns: byte[] or null	Returns the CCR record of a patient with an actor id matching the given patient id. If no such CCR exists on this CCRStore, null is returned.
getRecordByActorID Parameters: Actor ID (String) Returns: byte[][] or null	Returns the CCR records with at least one actor ID found in the actors section matching the given actor ID. If no such CCR exists on this CCRStore, null is returned.
getRecordByPatient Parameters: First name (String), Last name (String) Returns: byte[][] or null	Returns the CCR document for a patient with a matching first or last name. One of the parameters may be left as null to only search for a matching last or first name. If no such CCR exists on this CCRStore, null is returned.
getRecordByPerson Parameters: First name (String), Last name (String) Returns: byte[][] or null	Returns the CCR document where a person listed in the actors secection matches both the given first and last name. One of the parameters may be left as null to only search for a matching last or first name. If no such CCR exists on this CCRStore, null is returned
listRecords Parameters: None Returns: String[]	Lists the document ID of every record available to the consumer on the CCRStore.
listPatients Parameters: None Returns: String[]	Lists the patient ID of every record available to the consumer on the CCRStore.
listActors Parameters: None Returns: String[]	Lists the actor ID of every actor listed in records that are available to the consumer on the CCRStore.
listHeaders Parameters: None Returns: CCRHeader[]	Returns the headers of all patient records available to the consumer on the CCRStore.
deleteRecord Parameters: Document ID Returns: None	Deletes the record corresponding to the given document ID. An exception is thrown if the record cannot be found, cannot be deleted or this CCRStore does not support deleting records
addRecord Parameters: byte[] Returns: None	Adds a new record to the CCRStore. An exception is thrown if this store does not accept new records or the record can not be added.
listRefernces Parameters: Document ID (String) Returns: String []	Returns a list of documents referenced in the CCR matching the given document ID. Null is returned if no CCR document is found on the CCRStore matching the given document ID.
getReference Parameters: Document ID (String), Reference ID (String) Returns: byte[]	Attempts to retrieve a document referenced in a CCR document. If the document is unavailable or there is a problem retrieving the document an exception is thrown.
addReference Parameters: Document ID (String), bytes[] Returns: None	Attempts to add a supplemental document to a CCR. If editing records is not supported by the CCRStore, or there was a problem adding the reference an exception will be thrown

Table 1: CCRStore Primitive Operations.

Based on the primitives provided by Table 1, the HCX infrastructure can be used for several applications. The most prevalent being to act as the middleware between two or more existing EHR systems, allowing them to access and share records. Assuming the systems were on the same private network or cloud an adaptor CCRStore and consumer would be created for each EHR system implementing the given interfaces and providing a service which any CCRStore consumer could query for records. Another use case would be the creation of cloud based EHR repositories which provide their records through CCRStore based services. Such repositories cloud take full advantage of the scalability of the cloud to provide records to existing EHR systems that can easily meet any increase in demand by spawning new CCRStores and copy all or a subset of the records to the new store. HCX can also be used to connect EHR systems outside of its private cloud. Using bridges and virtual private networks (VPN), a large number of EHR systems form different healthcare providers cloud be linked together to provide a centralized source for accessing health records. This would be extremely useful in the case of health surveillance or in an emergency where physicians need immediate access to a patient's records that may be held by another healthcare provider. Client bridges allow for the creation of personal health record systems by simply extending a consumer to offer specific records over a web based connection. This consumer could simply format the CCR document into human readable html Web Page via using simple XSL transformations such as the one provided by the CCR Acceleration Resources Project (http://sourceforge.net/projects/ccr-resources/). This would at a minimum allow patients to view their own health care information that is currently being shared in the cloud. However, the use of VPN (e.g. Google GBridge (http://www.gbridge.com/)) will enable us to

create a point-to-point encrypted network which provides secure information exchange between authorized services and consumers across private and public clouds. Extensive ongoing research efforts has been made to create lakehead public cloud (clutch.lakeheadu.ca) based on Amazon Elastic Cloud EC2 Technology (http://aws.amazon.com/) with fifteen nodes (see Figure 7).

Figure 7: Lakehead University Clutch Public Cloud.

5. Conclusions

WHILE the HCX system described in this paper is far from complete, it is a start at creating a distributed, modular and scalable system for sharing health records over a private cloud. We showed how to build and integrate a composite application using the Eucalyptus and Apache CXF DOSGi open source frameworks for sharing CCR EHR records. The developed HCX prototype comprising of composite modules (distributed across the cloud) and can be integrated and function as a single unit. HCX allows adaptors and bridges to be created for existing EHR systems and repositories so that records can be exchanged through a standard interface and CCR record format. This is accomplished by building DOSGi based services and consumers made scalable through the cloud. Screenshots of the web based client bridge interface and the rendered CCR record are shown in Figures 8 and 9.

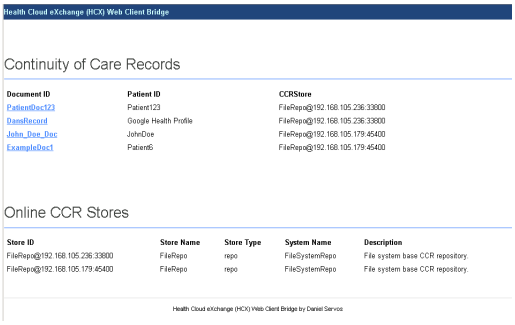

Figure 8: HCX Web Client Bridge

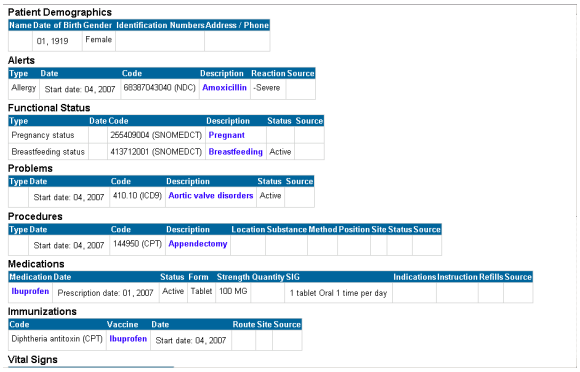

Figure 9: An Example of CCR Health Record.

There are many tasks left to our future research including providing higher information and data security as well as the connectivity between the HCX private cloud and the EC2 public cloud. That is besides using the integrated infrastructure for a vial healthcare application.

Acknowledgments: The authors would like to thank Amazon for their research grant and for allowing to us to use their public cloud space and services. Moreover the authors would like to thank Lakehead University Technology Service Center for providing the servers and computers required for our cloud computing research.

6. References

[1] Jeremy Geelan, "The Cloud Computing Ecosystem: The Top 100 Cloud Players" Journal of Cloud Computing, January 13, 2009.

[2] Robert Rowley, "Is cloud computing" right for health IT?", EHR Bloggers, August 6, 2009, www.ehrbloggers.com/2009_08_01_archive.html

[3] OSGi Alliance. (2010, Apr.) OSGi Alliance. [Online]. http://www.osgi.org

[4] Dion Hinchcliffe, Cloud computing: A new era of IT opportunity and challenges, March 3rd, 2009, http://blogs.zdnet.com/Hinchcliffe/?cat=34

[5] Health Level Seven International. (2010, Apr.) Health Level Seven International. [Online]. http://www.hl7.org

[6] N. E. M. Association. (2009) Digital Imaging and Communications in Medicine (DICOM): Part 1: Introduction and Overview. [Online]. ftp://medical.nema.org/medical/dicom/2009/09_01pu.pdf

[7] ASTM Subcommittee: E31.25, "ASTM E2369 - 05e1 Standard Specification for Continuity of Care Record (CCR)," *ASTM Book of Standards*, vol. 14.01, 2005.

[8] Care Management and Health Records Domain Technical Committee, "HITSP/C32: HITSP Summary Documents Using HL7 Continuity of Care Document (CCD) Component," *Healthcare Information Technology Standards Panel*, Version 2.5, 2009.