Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

User-controlled resource management in federated clouds

Marc Mosch[1,2*], Stephan Groß[1] and Alexander Schill[1]

Abstract

Virtualization and broadband Internet connections enabled what is known today under the term cloud. Benefits like scalability and cost reduction by pay per use agreements are accompanied by potential harm to the users data. Since existing cloud solutions can be considered synonymous with unknown locations and potentially hostile environments security protection objectives can not be guaranteed. Motivated by cloud's potential we propose a way to get rid of this drawback. We present π-Cloud, a personal secure cloud, that enables users to benefit from cloud computing and retain data sovereignty by federating the users own resources. More precisely we present a cloud resource manager that is able to keep control on the devices forming the user's π-Cloud as well as the services running on them.

Keywords: Cloud computing; Federated clouds; Cloud resource management; User-control

Introduction

Catchwords like scalability, on demand self service, pay per use, availability everywhere and any time describe an even bigger catchword: cloud computing. Cloud computing has evolved from technologies like utility and grid computing or the Internet of services. The progress in the area of virtualization and the availability of broadband Internet connections even for average consumers enabled cloud computing. It allows the outsourcing of computing and storage and involves the renting of virtualized hardware as well as of software running on it.

The cloud computing paradigm focuses on offering services and differentiates between three service levels [1]: As a foundation, the Infrastructure as a Service (IaaS) layer offers pure virtualized hardware like computing, network and storage. On top of this, the Platform as a Service layer (PaaS) provides a development platform for software which can then be utilised as Software as a Service (SaaS). Orthogonally to these service layers, four types of clouds are defined based on their consumer groups. The type mostly referred to when talking about clouds is the public cloud. Users of public clouds share the same resources under control of a cloud provider. In contrast, the users of private clouds are provided with resources that are maintained by themselves or at least for them alone. Hybrid clouds are a combination of public and private clouds while community clouds are a combination of several private clouds. Furthermore, we define federated clouds in contrast to other work such as [2] not only as synonymous to hybrid clouds but as a special mixture of hybrid and community clouds. To be more precise, a federated cloud is formed by an individual combination of public and private cloud resources as defined by an arbitrary participant of a community cloud.

With the outsourcing comes the cost reduction as it is no longer necessary to run data centres that are dimensioned for maximum peak loads and that run most of the time underutilized. However, these benefits are accompanied by a severe drawback that unsettles companies and prevents them from using cloud solutions: the decrease of data control. Data once outsourced is exposed to loss, abuse and manipulation. Cloud users are not able to determine where their data is located and who has access to it. The three security protection objectives availability, integrity and confidentiality are endangered in cloud environments. This is where our approach of a personal secure cloud sets.

The π-Cloud approach

The FlexCloud research group [3] aims to enable users to outsource their data and benefit from cloud computing

*Correspondence: marc.mosch@tu-dresden.de
[1]Technische Universität Dresden, Department of Computer Science, Chair of Computer Networks, D-01062 Dresden, Germany
[2]Technische Universität Dresden, Faculty of Civil Engineering, Institute of Construction Informatics, Dresden, Germany

without losing data control. This is achieved by dividing the used resources within a federated cloud into trustworthy and non-trustworthy ones. The devices under complete control of the user are per definition trustworthy whereas foreign resources are assumed to be non-trustworthy until further classification. This personal secure cloud, or π-Cloud is controlled by the so called π-Box, a security gateway that manages the separation of sensitive from public data. The former is stored preferably on trustworthy resources although it might be outsourced if necessary. An automatic encryption beforehand ensures data protection. Public data can be outsourced unencrypted. While this approach ensures integrity and confidentiality of important data the usage of information dispersal mechanisms ensures availability as well. See [4] for further details regarding the implementation of this feature. Similar mechanisms apply for the distribution of services. So services processing critical data should be bound to trusted resources only.

Thus, the π-Cloud's major objective is to put the user in a position to externalize his IT-infrastructure without losing the control over his data [5]. Therefore we form the π-Cloud consisting of all resources, services and data under the complete control of the user. The user is able to adjust his π-Cloud to his actual demands by secure inclusion of foreign resources. In doing so, data flow as well as service execution has to be controlled. Cloud setup and control, data flow as well as service distribution are regulated by the π-Box. The π-Box is composed of four main components, as depicted in Figure 1: (1) the Cockpit, (2) the Service Controller, (3) the Data Controller and (4) the Resource Manager.

The Cockpit provides the user interface. Although we are aiming to include as much intelligence in our π-Box as possible, to disburden the user from cumbersome administration tasks, it would be exaggerated to claim that the π-Cloud is able to maintain itself. In order to prevent the user from becoming the weakest link in the security chain, the cockpit has to put him into a position to supervise and manage his π-Cloud even if he is not an expert [6,7].

The Service Controller is responsible for secure service execution. As complete homomorphic encryption is not yet real-time capable, we realize secure service execution by decomposing services into critical and non-critical sub-services. Critical sub-services process high-confidential data and are executed strictly on trustworthy resources, whereas non-critical sub-services are allowed to compute on arbitrary resources.

The Service Controller's counterpart, the Data Controller, takes care of secure cloud storage. We split each file into several slices and place each slice encrypted and attached by an authentication code on a different resource. Since only a subset of these slices is required to restore the original information, a high availability is realized [4].

Last but not least, the π-Box Resource Manager is responsible for managing the set of all available resources and services.

Contributions and outline

In this work we focus on the development of this Resource Manager. The remainder of this paper is structured as follows. Before actually analysing necessary requirements for the design of the Resource Manager, we first discuss drawbacks of existing cloud resource management approaches. Going on, we present our three-fold design concept that covers service description as well as device and service coordination. We then present an overview of our prototype implementation and discuss first evaluation results. We conclude with a final discussion of our achievements as well as future work.

Our main contributions are:

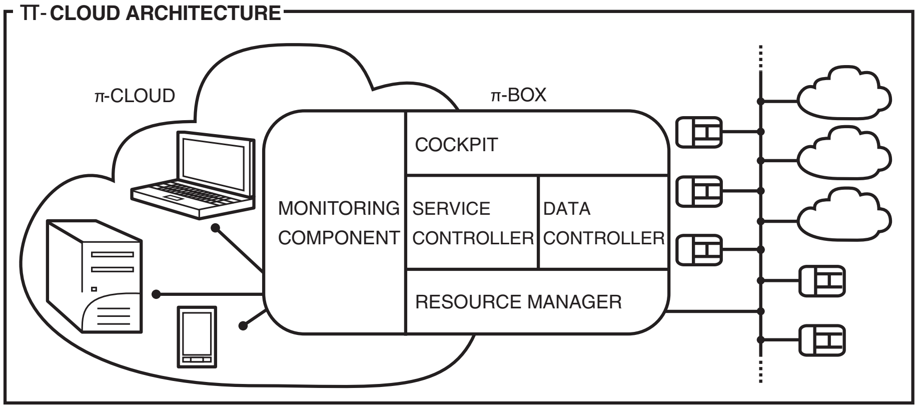

Figure 1 The π-Box. Architectural layout of the π-Cloud with a subdivision of the π-Box into Service and Data Controller, Cockpit, Monitoring Component and Resource Manager.

- The conceptional and technical development of fundamental system components for the setup of user-controlled federated clouds.
- This includes the development of CRDL, a Cloud Resource Description Language, that leverages the existing Open Cloud Computing Interface (OCCI) standard for the PaaS and SaaS layer.

A preliminary version of this work has already been presented at the Utility and Cloud Computing Conference in 2012 [8]. This revised version provides more details and insight in all aspects of our work. Furthermore, we have added two sections to discuss drawbacks of current cloud resource management solutions and to present first evaluation results of our Resource Manager prototype respectively.

Drawbacks of existing cloud resource management solutions

To gain an understanding, which functionality the Resource Manager has to provide, we start with an analysis of already existing solutions. In order to compare them with each other, we first of all define following reasonable evaluation criteria.

Availability as open source In order to prevent lock-in effects and to enhance security/trustworthiness, only Open Source solutions or such based on open standards are considered suitable.

Ability to integrate services from a user's devices Furthermore according to the π-Cloud idea presented in the previous section it is mandatory that the sought solution is able to integrate services from a user's own devices.

Ability to integrate arbitrary cloud providers The same applies for the integration of services and devices from different cloud providers. This includes community clouds.

Ad hoc migration of the managing component Additionally, we aim at enabling the ad hoc migration of the managing component itself between different parts of the π-Cloud due to stability, performance or trustworthiness reasons.

Support for IaaS, PaaS and SaaS Last but not least, our solution should support the whole bandwidth of cloud platforms, i. e. IaaS as well as PaaS and SaaS platforms.

For our investigation we concentrated on open source solutions and those proposed by academia. Industry solutions like *Akitio MyCloud* [9], *mydlink Cloud* [10], *Synology Cloud Station* [11] and *LenovoEMC Personal Cloud* [12] as well as *Samsung HomeSync* [13] and *myQNAPCloud* [14] have not been considered as they are proprietary, focussed solely on storage service and are bound to the respective companies' storage devices.

OwnCloud [15] is a promising open source solution which unfortunately does not support computing or platform services either and nearly no software services yet.

Most scientific approaches like the *Anzere* project [16], *PCP* [17] and *Cloud@Home* [18] are mainly storage centred, offer sometimes groupware functionality but not more and are therefore not suitable. Some scientific approaches like the *Clouds@Home* [19] project seem promising but are still work in progress.

Table 1 summarizes how the mentioned cloud resource management solutions match our evaluation criteria. Obviously, none of them fully meets our requirements which motivates the development of our own solution.

Requirements analysis

We start our requirements analysis by detailing the general tasks a cloud Resource Manager has to perform. Basically, these can be divided in three parts. On the on hand, the user's devices must be coordinated in order to combine them into a π-Cloud. On the other hand, there are the requirements from the other π-Box components. Specifically, the description of services to be run within the π-Cloud as well as their storage and management. Thus, we come to a high-level architectural overview of the Resource Manager presented in Figure 2.

The connector is more or less a straight forward component that encapsulates all necessary functionality for connecting the Resource Manager with the remaining parts of the π-Box as well as with the user's own devices or with external cloud resources. As it only contains technically state of the art mechanisms we skip a more detailed description here. Instead, we concentrate on the device and service coordination respectively.

Device coordination requirements

For coordinating the access to all relevant user devices in the π-Cloud the Resource Manager first has to recognize the (re)appearance of devices whose status has then to be maintained by a specific device directory. For this purpose, we distinguish between three general communication scenarios as depicted in Figure 3.

Intra-π-cloud scenario In this first scenario a resource wants to establish a connection to the π-box and both are inside a local area network. In such a situation as low operational effort as possible should be necessary. In ideal case the resource detection and interconnection between resource and π-box should work automatically. Furthermore the π-box should be safely identifiable and the communication should take place in a secure manner in order to prevent unauthorized access to the user's communication and data.

Table 1 Comparison of existing cloud resource management solutions

Approach	Open source	Integration of services from user's devices	Integration of arbitrary clouds	Ad hoc migration of the managing component	IaaS, SaaS and PaaS
Industry Solutions	No	No	No	No	No
OwnCloud	Yes	No	Yes	No	No
Sparkle Share	Yes	No	No	No	No
Aero FS	Yes	No	No	No	No
Anzere [16]	Yes	Yes	Yes	No	No
PCP [17]	Yes	?	?	?	No
[20]	Yes	Yes	No	?	?
[21]	Yes	Yes	No	No	Yes
Cloud CDI [22]	Yes	Yes	?	No	No
Social Cloud [23]	Yes	Yes	No	No	No
Cloud@Home [18]	Yes	Yes	Yes	No	?
Clouds@Home [19]	Yes	?	Yes	?	?
Cloud4Home [24]	Yes	Yes	Yes	No	No

The solutions are evaluated by means of the criteria defined above. If a matching is not certain due to no or imprecise information, a question mark is entered in the corresponding field.

Remote Intra-π-cloud scenario In this second scenario a resource tries to establish the connection to the π-box from the outside of the local network. Here it is necessary for that resource to know how to establish the connection to its own π-box. The requirements regarding identifiability and secure communication are just the same as in the Intra-π-cloud scenario.

Inter-π-cloud scenario If all the user's resources are registered – and with them the services – the user might want to use a service. In order to find an appropriate one he will send a query to the own π-Box. If the desired service is not available in the own repository the user can try to use the service of somebody else. Every π-Box runs its own repository. In order to use foreign services the π-Box has to connect to at least one other π-Box. This third scenario is the Inter-π-Cloud scenario where one or more π-Boxes interconnect to share information and resources.

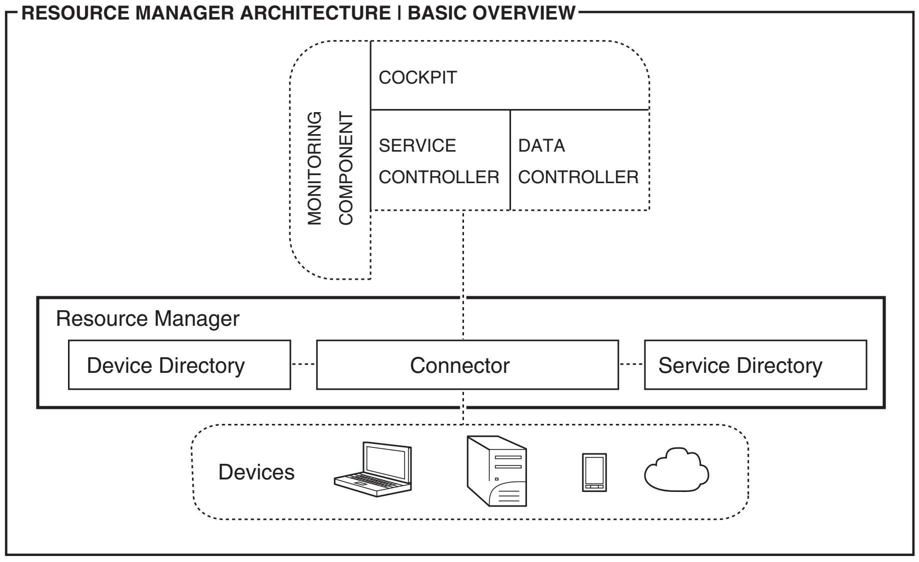

Figure 2 The Resource Manager. Basic overview of the Resource Manager's architecture.

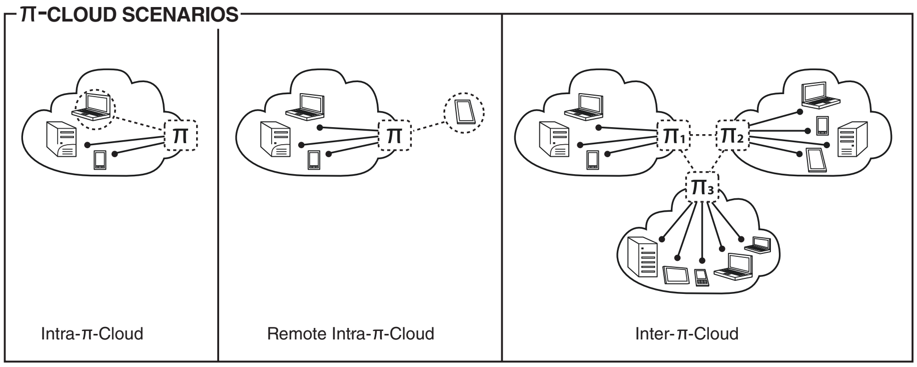

Figure 3 π**-Cloud communication scenarios.**

For each scenario the device directory has to be able to store all necessary information about available resources. It must be possible to search a device based on these information. Thus, the following two basic requirements are retrieved from the striven functionalities of the device directory.

Storing The information about devices might either be stored as a file directly into the file system or they might be stored in a storage system of any kind like for example in a relational database.
Searching Either a device initiates a search – for example to find the current π-Box – or other components of the π-Box incorporate the search function as a subroutine for other tasks. The result of a request is a set of attributes of the node.

Besides these functional requirements our system should also satisfy several non-functional requirements as follows.

Platform independence According to the π-Cloud idea every device might get the π-Box status which basically means that the same π-Box software has to run on all of them independent of their software and hardware platform.
Resource conservation To enable all devices to gain π-Box status, the Device Directory as well as the other π-Box components have to be lightweight enough to run – at least in small scenarios with a low double-digit number of devices – even on a smart phone if necessary without affecting its main functionalities. This means that a resource conserving architecture has to be chosen.
Security The device coordinator must support the user's wish for confidentiality, integrity and availability of his data when migrating it to the cloud. Therefore, strong security mechanisms must be integrated to protect the data traffic carried out by the device coordinator.
Scalability The number of the devices within the corresponding π-Cloud could vary from half a dozen in a home office to several hundred thousands in a big company. The device directory has to show a high scalability to manage such a large number of devices with satisfying performance.
Responsiveness Besides scalability another important aspect for the acceptance of such a system is its responsiveness. It is important to keep waiting times for search results as low as possible.

Service coordination requirements

Service description format requirements

During the handshake between device and π-Box the device has to publish its services. To describe them a service interface description format has to be found that is extensible, widely distributed, easy usable and that supports the description of non-functional properties. These requirements have to be meet for the following reasons:

1. *Extensibility:* For the description of cloud services they have to be differentiated based on the before mentioned service levels IaaS, PaaS and SaaS. Since PaaS and SaaS show a broad range of properties that differ from provider to provider and from service to service a fixed basic set of properties is not powerful enough to describe the services. For example the description of functional properties of a routing planner differ fundamentally from that of an office product. Even IaaS providers need the flexibility to extend the basic feature set. Although it might be assumed that a basic set of computing, storage and network properties is sufficient for them and that only units may change from time to time (the measure for computing power may for example change from GigaFlops to TerraFlops) a closer inspection shows that essential infrastructural changes occur. It just takes longer time periods for

changes that are fundamental enough to require the descriptiveness of new properties.
General-purpose computing on graphics processing units (GPGPU) is such a fundamental change. Powered by computing engines and interfaces like CUDA (Compute Unified Device Architecture) [25] and OpenCL (Open Computing Language) [26] users were provided with a huge performance boost that came with the utilization of GPUs (graphic processing units) for former CPU tasks. Amazon for example offers GPU computing instances since 2011 [27]. With this change came the need to extend the given set of property descriptions.

2. *Non-functional properties:* Given a set of services with similar functional properties the non-functional properties become important for the selection of the most appropriate service. That is why the description language has to be able to describe them as well. Non-functional properties include the functional description, costs, quality and safety.
3. *Distribution:* The distribution of the service interface description language plays another important role. In order to be able to integrate as much existing services as possible a widespread language has to be used.
4. *Ease of use:* Last but not least the ease of use is a major feature that should allow service providers to easily extend the basic set of property descriptions without being discouraged by to complex handling. Therefore the service interface description language has to be of as low complexity as possible while being as complex as necessary.

Service directory requirements

To manage all available services within the π-Cloud we further introduce a service directory. It has to be able to store service descriptions and extract information from them to build search indexes. It furthermore has to be able to deliver whole service descriptions or specific information about them if users request so. Hence, the following requirements are retrieved from the striven functionalities of the service directory:

Storing The service descriptions might either be stored directly into the file system or they might be stored as a string in a storage system of any kind like for example in a relational database. Furthermore the storage subsystem has to be able to store extracted elements and attributes in a way that future extensions of the service description format can be handled. For management reasons meta data like IDs have to be stored in the same place as the information extracted from the service descriptions.
Parsing After storing a new service description, the system has to extract relevant data from it. Due to the fact that it can not be foreseen which data might be requested by users and which not, all the elements and attributes have to be considered relevant. The whole content of the service description is therefore extracted and in the following referred to as relevant data. Since the service description format might be extended, the parser should be able to deal with new elements.
Searching It is not only the user that should be enabled to initiate a search. Other components of the π-Box might incorporate the search function as a subroutine for other tasks. The data controller might for example run a background task that searches for suitable storage services to disperse the data to. For a broad access to the stored information the ability to cope with syntax variety is important. The use of a lexical analyser that can for example handle fuzzy queries and replace synonyms can make the search for users more intuitive and flexible. π-Box components that have to access the service directory are better suited with a machine readable query language. To face this demand the service directory has to be provided with an interface that allows to couple a variety of query modules to it. The result of a request is however a list of suitable services. These might be ordered according to a rating either based on information from a monitoring system or on user decisions. Although the rating system necessary for this task is out of focus of this work, the service directory has to be designed open enough to be easily extended with such a system.
Retrieval If the user chooses one of the offered services from the list, additional detailed information and the whole service description might have to be retrieved. Authorisation and authentication rules have to be part of an other subsystem of the π-Box or should be encapsulated in a library that all π-Box components can use.

In addition, the same non-functional requirements already stated for the device directory apply here, too. Concerning the scalability a maximum amount of 500,000 devices for big companies was estimated. Assumed that not all devices provide services, an average of two services per device is likely which sums up a total amount of 1,000,000 service descriptions. With the amount of managed service descriptions the response time increases and it becomes increasingly complicated to keep users patient if the search is executed directly on the descriptions. So the responsiveness of the system has to be high enough to react within 3 to 4 seconds or faster [28]. For the adding of service descriptions this can be achieved with caching mechanisms if necessary. In contrast the retrieval of information has to rely on an efficient indexing technology, e. g.

with binary search trees. Since service descriptions can describe different resources, they contain various, sometimes unrelated data. That implies the need for multi tree indexes. A search combining different parameters results in the utilisation of the same amount of search trees. There is the need for a manager for different binary trees and for the trees themselves. But instead of creating them from scratch it seems to be more economically to consider only such existing storage systems that are able to generate the trees and manage them. The storage system therefore has to analyse the service descriptions and generate binary search trees based on extracted elements and attributes. Furthermore it has to have mechanisms to organise the search over multiple trees.

Designing the resource manager

In the following we discuss the conceptual design of the Resource Manager based on the requirements determined in the previous section.

Device coordination

The design of the device coordinator architecture is presented according to the three scenarios introduced in the last section.

Intra-π-Cloud In the previous section low operational effort was defined as a requirement when a resource wants to establish a connection to the π-Box in a local network. During this initial handshake phase resource and π-Box meet each other for the first time. The resource shares information about its available services and gets an ID. For the automatic resource discovery Zero Configuration Networking (Zeroconf) [29] seems promising since it is a configuration-free network approach that proposes techniques to build up an Internet Protocol without intervention from human operators. Participating resources can automatically connect to the network and get network addresses assigned. The two most common implementations are Bonjour [30] from Apple and Avahi [31]. Bonjour is an implementation not limited to Apple OS X and iOS. It also works on Linux and Microsoft Windows. Unfortunately Bonjour is only partially open source under Apache 2.0 license and partially closed source under an Apple license. We aim to offer the π-Box as open source. That is why we prefer Avahi which is an open source implementation developed under GNU Lesser General Public License. The needed identifiability could for example be ensured with a PIN code that is shown at a diplay at the π-Box. Other ways to safely identify the π-Box involve for example certificates and a Public Key Infrastructure (PKI) to check these certificates or recommendation or reputation systems might be utilized. A hybrid encryption will ensure a secure communication that is more efficient than a asymmetric one. Therefore the π-Box first sends its public key to the resource. In case of an successful identity check the resource generates a symmetric key, encrypts it with the π-Box's public key and hands it over to the π-Box. Which for its part decrypts the symmetric key with the private key only known to the π-Box. After this the whole communication (which includes the remote Intra-π-Cloud communication) can take place encrypted in a lightweight manner with the symmetric key that is now only known to the resource and the π-Box. Since we are in an early state regarding the communication protocol we can not offer deeper conceptional insight or implementations. The essence is that Zeroconf would enable automatic interconnection between resource and π-Box as desired and that the identifiability might be ensured via PIN code, PKI or recommendation or reputation systems while the secure communication should be realized with a hybrid encryption approach.

Remote Intra-π-Cloud In general it can be assumed that a remote connection to the π-Cloud follows an initial resource registration like discussed in the Intra-π-Cloud scenario. If so the necessary information like a constant IP and a port to contact the π-Box from the outside were already handed over to resource by the π-Box. If not for example because the user got in possession of a new device and is eager to test it before he enters his home network he has to know these information from memory. The solutions for identifiability and secure communication can be based on the solutions for the Intra-π-Cloud scenario.

Inter-π-Cloud The Inter-π-Cloud scenario is conceivable in two forms. Either all π-Boxes are part of a friend of a friend (FOAF) network. In this case it can be assumed that the users π-Box is in possession of the connection information to all the users friends' π-Boxes. Then a directed multicast might be the best way to query for a desired service. Then again if it is assumed that the user does not have such a FOAF at least one other π-Box has to be known. In this case a structured peer-to-peer network can be the solution. If a user enters such a network where he only knows one other π-Box the communication is not limited to the known instance. In fact directly addressed π-Boxes should be able to forward failed service queries or to introduce other instances to the new π-Box. To cope with π-Boxes that are leaving the network unattended because of faulty internet connections or hardware failures the network has to be robust. That is the network should be able to heal itself and replace information that are missing due to the unannounced absence of the peer. Furthermore it should be fault-tolerant and highly effective to ensure successful routing of messages.

Given these scenarios, we have identified several options for the general architecture of the device directory as depicted in Figure 4.

Going on, we analysed the use cases for the Intra-π-Cloud scenario (see Figure 5).

Join π-cloud The first use case captures the contact initiation between a device and the π-Box. A successful attempt results in the π-Box revealing itself and sending information to the device how to connect from the outside of the personal network. Furthermore a certificate has to be generated and handed over to the device to enable it to identify itself as an authorised member of the π-Cloud.

Create connection This use case deals with the creation of a secure connection between a device and the π-Box. It is required that the π-Box is clearly identifiable and that the communication takes place in a secure manner in order to prevent unauthorised access to the user's data. The device sends information about itself to the π-Box. After a authorisation check the π-Box generates a session key for the ongoing communication. Afterwards the Device Directory marks the device is as connected and stores the session key.

Delete device record If a device has to be finally dismissed from the network – for example because it is broken – it has to be deleted from the list of managed devices in order to keep the data base up-to-date and slim.

Set disconnected For management tasks it is sometimes necessary to know if a device is available or not. Therefore it must be possible to set an entry in the data base that marks a device as disconnected.

Set π-box info The π-Box software should potentially run on all π-Cloud devices. The π-Box status can change from one to another. This means that the Resource Manager has to provide a way to assign π-Box status to a specific device.

Get π-box info If a member of the π-Cloud wants to contact its π-Box it has to be provided with a method to get to know which other member of the π-Cloud is the current π-Box.

Revoke certificate It has to be ensured that certificates once handed out by the π-Box can be revoked to exclude devices from the π-Cloud if they are responsible for access violations or other harmful behaviour. The revoking has to be initiated by a component of the π-Box which sends the ID of the device that has to be excluded from the π-Cloud. After the Device Directory added the certificate of the corresponding device to the revoke certification list, a list of the connected devices is requested which leads the Device Directory to return a list of them. A request should then be send to all devices of the list

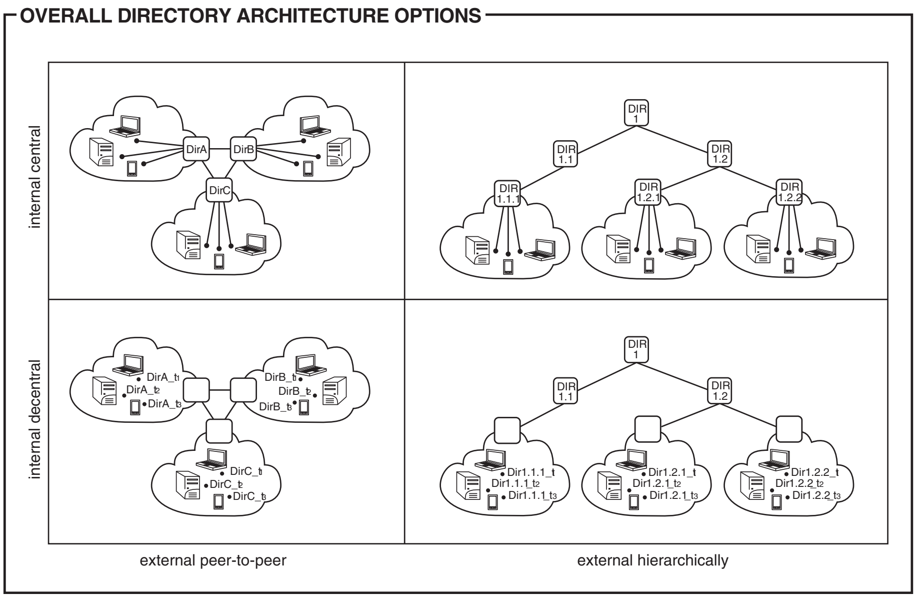

Figure 4 Architectural options of internal and external π-Cloud device directories.

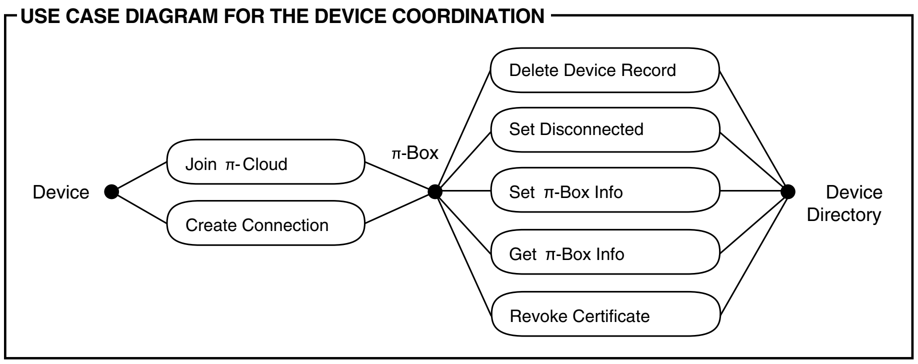

Figure 5 Use cases regarding the device coordination.

in order to cause each of them to add the undesired device to the own revoke certification list.

Service coordination

Service description

As the analysis shows there are four main features the service interface description language has to provide. It has to be easily extensible and support the description of non-functional properties while being widely distributed and of low complexity in order to achieve a good ease of use. Existing service interface description languages are either easily extensible and support the description of non-functional properties (like USDL and OWL-S) or they are widely distributed and show a good ease of use (like WSDL, WADL) as to be seen in Table 2.

To the best of our knowledge there is no approach that is fulfilling all four requirements in one solution. Here is where a meta model comes in hand that was designed to describe cloud resources. It is the so called Open Cloud Computing Initiative (OCCI) [32] powered by the Open Grid Forum [33]. At the moment it consists of a core model [34] and an infrastructure extension [35]. A combined diagram of both models is depicted in Figure 6. The graphic except the grey parts represents the OCCI core model with an infrastructure extension according to the specifications v1.1. Among other things this modularisation makes the model easily extensible. Well known open source cloud attempts like OpenNebula, Openstack and Eucalyptus already implement OCCI [36]. Given this high distribution and the simple but easily extensible model OCCI seems to be an appropriate base for an own implementation to describe services.

Service directory

The design of our service directory is based on the use cases depicted in Figure 7.

Add service description To be able to describe cloud resources, CRDL, a Cloud Resource Description Language, was developed. It leverages the existing Open Cloud Computing Interface (OCCI) standard for the PaaS and SaaS layer. To publish the services of a device an appropriate CRDL file has to be added to the Service Directory. Therefore the file is sent to the to the Resource Manager. The Resource Manager checks the file's validity and adds some meta data like the user's id. The file is then translated in a format that can be understood by the Resource Manager's Service Directory where it is sent next. In the Service Directory a file id is generated and added as meta data. Furthermore the CRDL file has to be parsed to extract relevant data. All information will then be stored in the Service Directory's own storage. Finally an acknowledgement and the newly generated id will be sent back to the device.

Search service To search a service a device sends a respective search request to the π-Box. The request is handled by Service Directory's Storage module. As a result the Service Directory returns a list of CRDL files matching the search criteria. The Resource Manager as well as

Table 2 Comparison of service description formats

	USDL	WSDL	WADL	OWL-S	WSMO	OCCI
NFP Support	Very good	Existing	Existing	Existing	Existing	Existing
FP extensibility	Existing	No/bad	No/bad	Good	Good	Very good
NFP extensibility	Very good	No/bad	No/bad	Very good	Very good	Very good
Distribution	No/bad	Very good	Existing	No/bad	Existing	Existing
Ease of use	Existing	Very good	Very good	No/bad	No/bad	Good

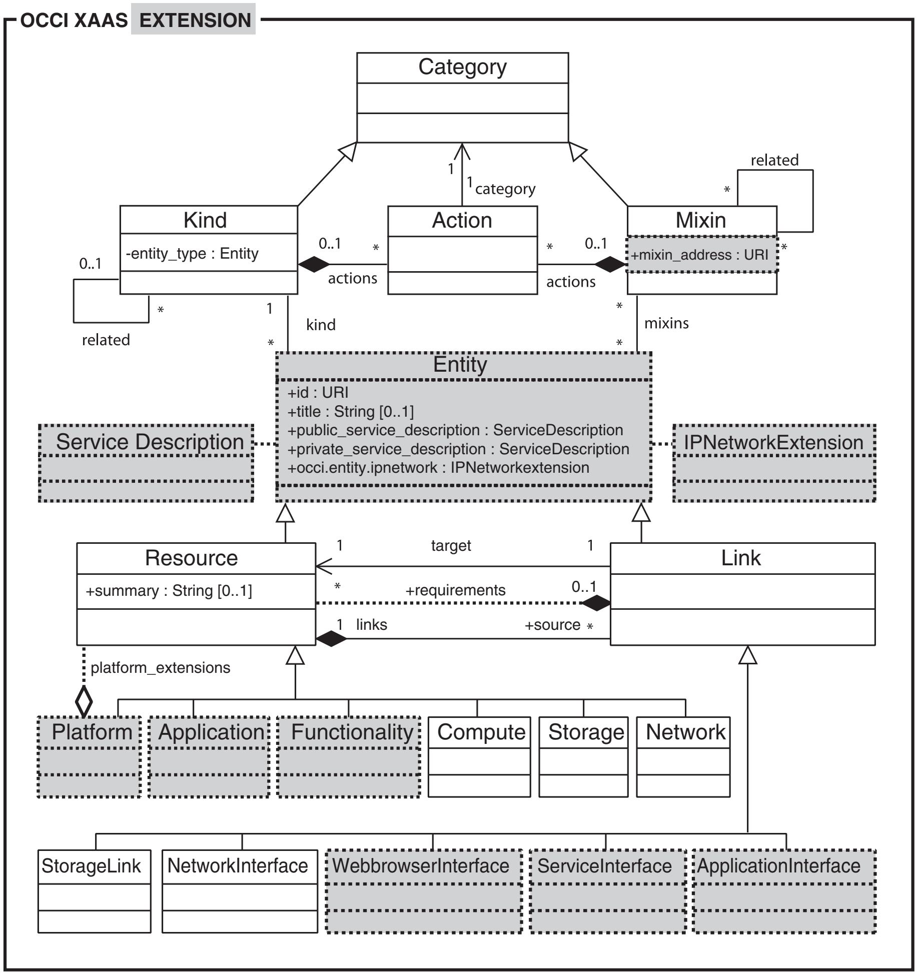

Figure 6 OCCI extensions. A simplified representation of the OCCI Core Model with infrastructure extension for the IaaS domain according to the specifications v1.1. The grey parts mark an additional extension for the integration of existing service description formats as well as additional extensions for non-functional properties, platform and software extension for the XaaS domain and required interface extensions.

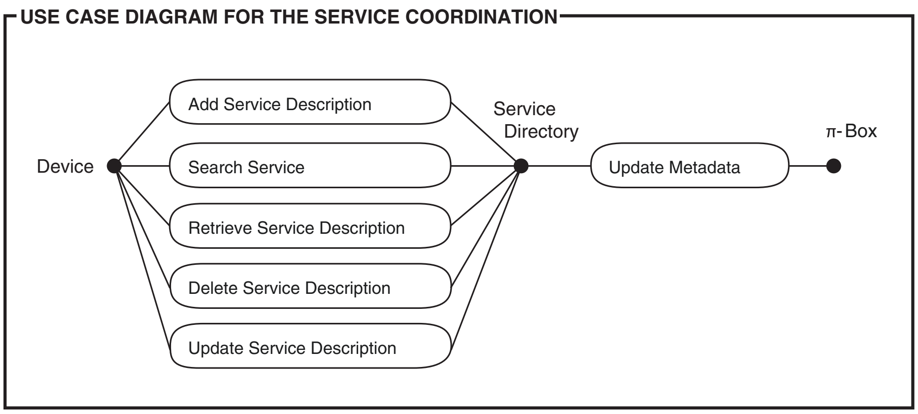
USE CASE DIAGRAM FOR THE SERVICE COORDINATION
Device
Add Service Description
Search Service
Retrieve Service Description
Delete Service Description
Update Service Description
Service Directory
Update Metadata
π-Box

Figure 7 Use Cases for the service coordination.

other components of the π-Box can filter and reorder the results, for example according to the knowledge of a rating module, before they are resend to the inquiring system.

Retrieve service description Retrieve a CRDL file based on the ID assigned during the addition of the service description.

Delete service description Delete a CRDL file based on the ID assigned during the addition of the service description.

Update service description Update a CRDL file based on the ID assigned during the addition of the service description.

Update metadata Update a CRDL file's metadata based on the ID assigned during the addition of the service description.

Overall architecture

A schema of the Resouce Manager's architecture is shown in Figure 8. It consists of three main modules, one for each directory and one – the Connector – as an interface. This Connector itself again contains three modules. The Internal Call Manager is responsible for the communication with other π-Box components while Listener and External Call Manager cover the communication with external resources. The Device Directory consists of two modules. The Core module encapsulates management functionalities and the Storage module offers storage functionalities. In addition to two similar modules, the Service Directory contains a parser module, which is responsible for the parsing of the CRDL files.

Prototype implementation

Device directory

According to the π-Cloud idea, the π-Box software is intended to run on a variety of hardware platforms. That is why the prototype is based on the platform-independent programming language Java. With future migration in mind, devices and π-Box should run different instances of the same program. Migrating the π-Box from one device to another then just involves a change of the status of both devices and a transfer of administrative tasks, information and rights. It seems appropriate to use RMI the Remote Method Invocation protocol for the communication. It is Java's version of an RPC (Remote Procedure Call). An RPC enables one program to execute procedures in the foreign address space of another program on a different machine. The Java RMI API allows to invoke Java methods remotely from another (JVM) (Java Virtual Machine). The machine which is in possession of the remote object registers it at the RMI-registry with a unique name. The machine that strives for remote access uses the object's name to retrieve an object reference from

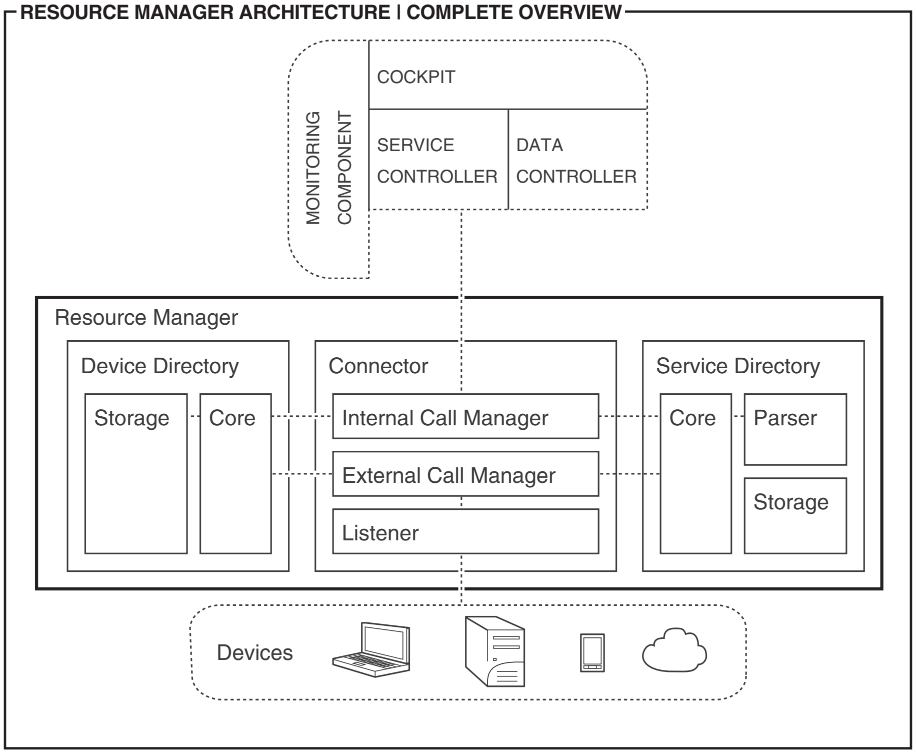

Figure 8 **Complete overview of the Resource Manager's architecture.**

the RMI-registry. The object reference has to correspond with its remote interface. If the machine that strives for remote access invokes a method from the object reference of the machine which is in possession of the remote object, the method on the remote object is executed. In this process external code might be loaded dynamically for usage. As a result of the invocation return values or an error message are sent back. This response is useful in the communication process between client and π-Box since in most cases it is initiated by the client which relies on a response. In seldom cases the communication is initiated by the π-Box and is then sent to all connected clients in the network. But this is a rare event. It is so rare, that RMI overhead compared with publish subscribe mechanisms is acceptable.

The most obvious storage solution for data sets is storing them in the file system. But if the data to be stored are well structured, data bases can be a faster solution. Data bases that are able to store the data completely in main memory have to be preferred. They utilise high throughput and low response times of main memory and achieve a much better performance than common data bases, which work mostly on comparatively slow hard disks. With HSQLDB (HyperSQL DataBase) [37] one of the most popular Java-based open source databases is used which is able to run completely in-memory.

Service description

The base of the π-Box service description is the OCCI core specification with some extensions implemented as XML Schema. The class resource is complemented by a complex data type converter specifications that consist of:

- a classification of the service description of the service provider as Enum (e.g. WSDL,WADL and OWL-S)
- the corresponding service description address in form of an URI
- the address of the converter for the extraction of the needed data from the service providers service description in form of an URI
- an execution instruction for the converter (e.g. XSLT for WSDL/WADL/OWL-S converter in XSL)

This converter specifications are stored in a separate local file and can therefore be easily extended by the user for other forms of service descriptions. The core class Link is extended by inheritance for diverse interface types of the services in addition to existing specifications of the OCCI infrastructure model. The resulting interface class structure is integrated into the resource class as an abstract element. Its ascertainment of goods can be generated at runtime – together with other resource elements – from the providers service description by means of the converter specification. The class resource uses the extensions for infrastructure services of the OCCI infrastructure model. With Software and Platform own extensions for software and platform services are added in form of ascertainments of goods of the resource class. The class Platform is composed of one or more instantiations of each of the following: Compute, Storage and software. Non-functional extensions (e.g. for quality and safety features) as well as individual extensions of the service description by the π-Box users are integrated via Mixins. Mixins allow to extend Ressource with additional arbitrary attributes. For reasons of manageability and clarity of the service description the class Mixin is extended by the address of the Mixin in form of the URI of the XML source file which is integrated at runtime.

Service directory

For the implementation of the service directory we have existing basic technologies that are open source and therefore compatible with the π-Cloud idea. Table 3 summarizes the results of our analysis. We have chosen Lucene as our founding implementation component since it fulfils all our functional requirements. The fulfilment of non-functional requirements is only of interest for the complete Service Directory. For a basic technology like Lucene it is only important to be platform independent, resource conserving, scalable, responsive and able to cope with CRDLs inherent complexity. According to the Apache project [38] all these requirements are fulfilled.

Evaluation and discussion

In this section we describe our evaluation of the Resource Manager's efficiency. After introducing the evaluation methodology, our results are presented and discussed.

Table 3 Comparison of basic technologies for the service directory

	LDAP	RDBMS	File system	XML Query Languages	Digester	Lucene
Parsing	Impossible	Impossible	Impossible	Intended	Intended	Intended
Storing files	Possible	Intended	Intended	Impossible	Possible	Possible
Storing data	Intended	Intended	Possible	Impossible	Possible	Intended
Searching	Intended	Possible	Possible	Impossible	Impossible	Intended
File retrieving	Intended	Intended	Impossible	Impossible	Impossible	Intended

Methodology

In order to validate sufficient performance in real-world scenarios, several test scenarios are applied. As already discussed the π-Box and thereby also the Resource Manager have to be scalable to manage even big usage scenarios with up to 500,000 devices and 1 million CRDL files. At the same time the response of the system should not exceed 4 seconds to provide a good user experience [28]. The tests cover this big scenario with 1 million CRDL files and a middle size scenario with 100,000 CRDL files. Even smaller scenarios are indirectly included in the middle size scenario.

Service Directory and Device Directory are accessed by only one Connector and therefore have to handle requests only sequentially. The Connector can be accessed by several devices at the same time. Therefore in theory it has to manage parallel requests and is responsible for the execution order. However, the prototype takes no care of this. Since for the tests only one client sends requests they enter the connector one by one and are executed sequentially. So the scalability can just be assumed.

Four test parameters can be used to judge the performance of the developed Resource Manager. The parameter which represents the efficiency of the Resource Manager the most is the *response time* (t_{resp}). It is defined as the period between the moment when the device sends the request and the moment when it receives the response. The period starting with the point when the Listener module receives the query and ending when it sends the response is the period which defines the execution time. Thus, the response time equals the execution time (t_{exe}) plus twice the network delay (t_{nd}):

$$t_{resp} = t_{exe} + 2t_{nd}$$

Another important parameter is the *CPU load*. As already mentioned the π-Box may run on a device which has another original purpose. If a smart phone acts as a π-Box, it should for example still be able to make and accept calls. That is why the CPU should be used economical – at least in smaller π-Clouds. In case of bigger π-Clouds, it seems plausible that the π-Box is hosted by a dedicated server, rendering CPU load less relevant, as long as it is not excessive.

The *memory consumption* is a further parameter which may also interfere with using the host system in smaller scenarios for its original purpose. It is hard to find an objective requirement regarding the acceptable amount of utilised memory. That is why it is only possible to rely on subjective estimation.

Last but not least the *size* of the Device Directory's *database* and the Service Directory's *index* are important parameters. They have to be within reasonable limits.

Our **test cases** have been designed with all processes in mind which involve a waiting user. These are:

- join π-Cloud
- create connections
- add new CRDL files
- search services
- retrieve CRDL files
- delete CRDL files
- update CRDL files

Furthermore these processes have to be evaluated under different sizes of π-Clouds according to the before mentioned usage scenarios. The evaluation covers a middle size π-Cloud with 0 to 100,000 CRDL files and 50,000 devices and a big π-Cloud with 10,000 to 1,000,000 files and 500,000 devices.

To reflect conditions of real world personal clouds, the client server communication takes place inside a local network. Apart from that, the systems differ for the two π-Cloud sizes. The middle size scenario involves a laptop computer with the following characteristics hosting the π-Box:

- Model: Lenovo G550
- CPU: 2.1 GHz Intel Core 2 Duo
- Memory: 3072 MB RAM
- Architecure: 32 bit
- HDD: WesternDigital 500 GB 5400 rpm (ATA WDC WD5000BEVT-22ZAT0)
- OS: Debian 6.7 Squeeze
- Runtime Environment: Oracle JDK 7

The laptop computer itself was the server and a QEMU-KVM 0.12.5 based virtual machine with 1 CPU core and 128 MB of RAM was used as client. The server for the big scenario is a virtual machine with the following specifications:

- CPU: 4 × 2.4 GHz (only one core used)
- Memory: 8192 MB RAM
- Architecure: 64 bit
- OS: Debian 6.7 Squeeze

The client virtual machine shows the following characteristics:

- CPU: 1 × 2.0 GHz
- Memory: 500 MB RAM
- Architecure: 64 bit
- OS: Debian 6.7 Squeeze

Both virtual machines run on a host system with the following characteristics:

- Model: Fujitsu Primergy RX300S6
- CPU: 2 × Xeon E5620 2,4 GHz 4C/8T 12 MB
- RAM: 4 × 12 GB
- HDD: Fujitsu ETERNUS DX APAK 6x750 GB (Raid5)
- Virtualisation Environment: VMWare vSphere 4.1

The measurements presented in the following should provide a coarse comparison of the performance of the two systems. They were taken with the GNU/Linux file copy and conversion command `dd`.

The CRDL files for the test were generated based on four main structure types – one for storage, one for compute, one for platform and one for software descriptions. To achieve good diversity which means a huge set of unique CRDL files, parameters were changed randomly within each of the four groups – always within predefined limits.

The queries were generated at runtime. There are four query types: the fastest possible without parameters which retrieves all services; the normal one, a query for storage, with different sizes and status online; the slowest, a query for compute services with close restrictions for all five possible parameters; a query which searches for storage with unsatisfiable demands.

Results

Response time

Join π-cloud In the middle size scenario run on the laptop computer the fulfilment of join requests on server side (execution time) took 2.25 milliseconds in average with a maximum of 202 milliseconds. 98.9 percent of the request where processed within 2 to 8 milliseconds. And 99.9 percent took not longer then 64 milliseconds. The overall network delay with an average of 2.25 milliseconds leads to an average response time of 4.89 milliseconds and a maximum response time of 206 milliseconds on client side. 98.2 percent of the responses reached the inquiring client within 2 to 8 milliseconds. And 99.9 percent took not longer then 68 milliseconds. Since the overall network delay is constant and negligible small the execution time on server and the response time are almost identical. We found that the response time is independent from the amount of already joined devices. This result was expected. It reflects the fact, that information about already registered nodes is not retrieved during the join process.
In the big scenario executed on the virtual machine the fulfilment of join requests on server side took 1.04 milliseconds in average with a maximum of 2179 milliseconds. The second longest join was executed within around 1 second. 99.994 percent of the request where processed within less then 10 milliseconds. And only 10 of 500,000 joins took longer then 100ms. The overall network delay with an average of 1.04 milliseconds added to a total average execution time of 1.73 milliseconds. 99.988 percent of the responds reached the inquiring client within less then 10 milliseconds. As in the mid-size scenario, we found that the response time for join requests is independent from the amount of already joined devices.

Create connections In the middle size scenario the response time grows almost at a linear rate with the amount of registered devices. After each 10,000 join operations the time for connecting the according clients was measured. After joining the last 10,000 clients to the π-Cloud each of the subsequent connections took around 38.5 milliseconds in average from sending the request to receiving the acknowledgement. The drop at the beginning is the result of initial just-in-time compilation of the JVM. In the big scenario the response time also grows almost at a linear rate with the amount of registered devices. After each 100,000 join operations the time for connecting the according clients was measured. After joining the first 100,000 clients to the π-Cloud each of the subsequent connections took around 34 milliseconds in average while the last 100,000 connections took around 264 milliseconds from sending the request to receiving the acknowledgement.

Add new CRDL files In the middle size scenario run on the laptop computer the fulfilment of an add request on server side took 262 milliseconds in average with the four slowest responses between 6 and 11 seconds. 88 percent of the requests were answered in less than 0.3 seconds and 99.8 percent took less than 2 seconds. Since the overall network delay is similar to the delay in the join case the execution time can be considered almost equal to the response time.
In the big scenario, run on the virtual machine the fulfilment of an add request on server side took 29 milliseconds in average which is almost ten times faster than on the laptop computer. The four worst results reached from 20 to 45 seconds. Nevertheless, 99.83 percent of the requests were answered within less than 300 milliseconds and 99.99 percent within less then 1 second. Since the overall network delay is similar to the delay in the join case the execution time can be considered almost equal to the response time.

Search services As it can be seen in Figure 9 the response time for search requests depends on the size of the index and grows with a linear rate.

Retrieve CRDL files After each 1,000 added CRDL files the retrieval of 100 files was measured. The retrieval took an average of 6.7 milliseconds. The slowest request took 71 milliseconds. 99 percent of the requests took not more than 14 milliseconds. 99.9 percent of the retrieve requests took less than 39 milliseconds and 99.99 percent less than 45 milliseconds. We found that the response time grows only negligibly for the mid-size scenario.

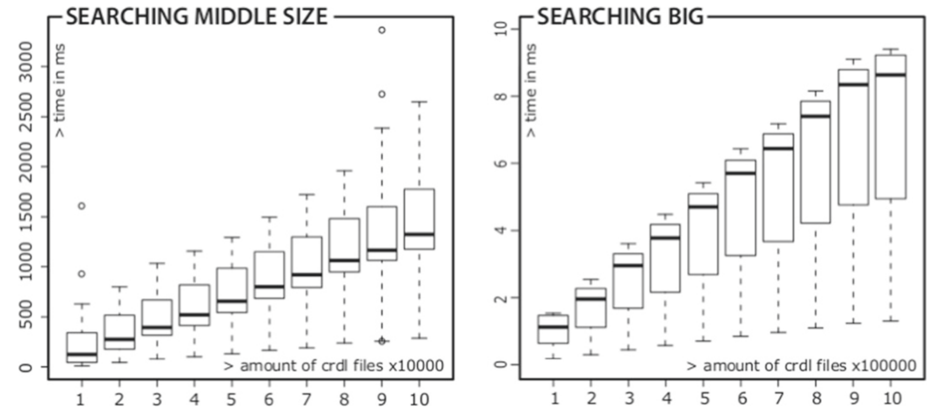

Figure 9 Response time for the search a service request. The abscissae show respectively the amount of already added CRDL files and are therefore like time-lines. The ordinates both show the response time in milliseconds. For the response times for every 10,000 requests maximum, 75%, median, 25% and minimum are shown. The circles mark faulty measured values not considered for the determination of these quantiles.

After each 10,000 added CRDL files the retrieval of 1,000 files was measured. The retrieval took an average of 9.3 milliseconds. The slowest request took 40 milliseconds. 99 percent of the requests took not more than 14 milliseconds. 99.9 percent of the retrieve requests took less than 28 milliseconds and 99.99 percent less than 37 milliseconds. Just as in the mid-size scenario the response time for the big scenario grows negligibly.

Delete CRDL files After each 1,000 added files 10 of them were deleted for this test. The response time for the deleting requests took 121.9 milliseconds in average. 90 percent of the requests took not longer than 146 milliseconds. The longest request took 302 milliseconds. Up to 100,000 added files the response time grows with the size of the index – but not considerably.

After each 10,000 added files 10 of them were deleted for the big scenario test. The response time for the deleting requests took 20.5 milliseconds in average. The longest request took 331 milliseconds. 90 percent of the requests did not take longer than 25 milliseconds. Up to 1,000,000 added files the response time grows with the size of the index. Nevertheless the response time is superior considering that the average value is in the range of black-white-black response times of LCD displays. And even the worst result with 331 milliseconds corresponds with the duration of a blink of the human eye.

Update CRDL files For this test 10 files were updated every 1,000 added files. The response time for the updating took 371,6 milliseconds in average. The longest update took 2,773 milliseconds. 90 percent of the requests were answered within a maximum of 438 milliseconds. The growing of the request time with the amount of files in the index is negligible.

The same applies for the big scenario where 10 files were updated every 10,000 added files. The response time for the updating took 55.2 milliseconds in average. This is almost 7 times faster compared with the middle size scenario – the result of the more powerful host system. The longest update took 780 milliseconds. 90 percent of the requests were answered within a maximum of 79 milliseconds.

Scalability assumptions It can be assumed that the response time will increase with the number of parallel requests. Given the almost consistent low response times for the different processes this increase can be assumed marginal. Except for the search process, it is most likely that the response time will not exceed the critical 4 seconds mentioned before. However, referring to this 4 second threshold, the search process only performed well for up to 300,000 stored service descriptions. The number of manageable descriptions will decrease with the growing number of parallel requests, since the regular expressions used for the search are very resource intensive. However, it is likely that companies big enough to depend on hundreds of thousands of service descriptions will have powerful dedicated π-Box servers to speed up the response times to a bearable extent. Furthermore it has to be pointed out that the developed architecture only covers service brokering. Regarding performance the service brokering is far less important then the actual service usage. The usage is responsible for the main traffic and takes place between the clients only. No server is involved.

Index and database size

The generated CRDL files have a size between 2 KB and 9 KB with an average of 6 KB. Figure 10 shows that the

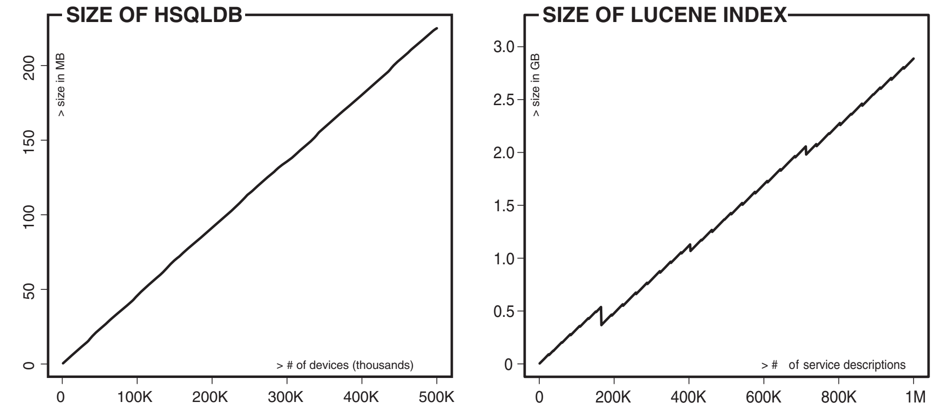

Figure 10 Lucene index and HSQLDB data base size in relation to the number of service descriptions and devices.

size of the index grows at a linear rate with the amount of files. The index including CRDL files plus extracted data and meta data is 1.6 to 1.9 times smaller than the storage space needed by the original CRDL files. 590 MB for 100,000 CRDL files mean a Lucene index of 361 MB. In the big scenario the compression is even more efficient most likely due to a higher rate of reusable patterns for Lucene's compression algorithms. With growing index the compression induced perennial index size reduction shows as a spiky graph. The storage space of 5,896 MB for 1,000,000 CRDL files is reduced to 3,019 MB if the files are stored and described in the index. Extensions of the description via meta data will increase the size of the index but the growing will be negligible. The size of the database also grows at a linear rate. 50,000 joined devices occupy 22.6 MB if they are all connected. 500,000 devices use 219 MB. The size of the data base depends on the number of joined and connected nodes and on the amount of revoked certificates.

CPU load and memory consumption

Table 4 shows CPU load as well as memory and hard disk consumption for the different use cases. The adding process depends more on the hard disk than on the CPU or the memory. On the laptop computer there were for example only 10 to 20 percent CPU usage. The comparison of the adding process for laptop computer and virtual machine shows that a fast hard disk can shift the bottleneck. Since the virtual machine's host was a server with 6 fast SAS disks in RAID5 mode the CPU became fully utilised. In contrast searching requests demanded full processing power on both, server and laptop

Table 4 Utilisation of processor, memory and hard disk for the specific use cases

Use case	CPU	RAM	HDD
Laptop computer			
Join and connect	60%–70%	78 MB–84 MB	Low load
Add	10–20	Grows linear	High load
Search	100%	330 MB–360 MB	Low load
Retrieve	100%	ca. 300 MB	Low load
Delete	20%–30%	ca. 200 MB	High load
Update	30%–50%	ca. 350 MB	High load
VMWare			
Join and connect	73%–75%	140 MB–190 MB	Low load
Add	100%	Grows linear	High load
Search	100%	up to 2 GB per request	Low load
Retrieve	100%	up to 2 GB per request	Low load
Delete	up to 100%	up to 1.5 GB	Medium load
Update	50%–60%	ca. 400 MB	Medium load

The values for processor and memory are based on observations during the evaluation while the content of the last column is based on theoretical assumptions.

computer. The memory consumption did not exceed 360 MB on the laptop computer while the virtual machine had to provide the π-Box with up to 2 GB per request when the index held 1 million service descriptions. After the request the size of the program in memory fell below 200 MB.

Summary

The developed prototype works well for networks with a file amount of up to around 300,000 CRDLs with the given virtual test server and with a sequential order of requests. Larger amounts of service descriptions lead to response times for search requests of 4 seconds and more, at least with the given virtual test server. Anyhow, is likely that such large π-Clouds are managed by powerful dedicated π-Box servers, which would be capable of handling more service descriptions. Then again, the tested program was only a prototype. Not all designed features were included. The communication channel was for example not encrypted and was furthermore based on a local area network. Additional encryption will add overhead and increase the response time. Communication over the Internet can be substantially slower then our LAN-based tests, depending on the quality of the user's connection. But there are also possibilities to enhance the prototype with regard to its performance. The Lucene based Storage module of the prototype is single threaded and therefore only utilises one core. But since the design of the Resource Manager is modular, Lucene could be exchanged for example with ElasticSearch [39] – an extension of Lucene which enables distributed indexes. This would allow multi threading and therefore decrease the response time with the growing number of distributed indexes. The response time for the search requests could also be reduced if regular expression queries would be replaced by less resource intense query types. During the parsing process frequently searched properties could be extracted and added as meta data to the service records. Since this would prevent browsing indexes for whole CRDL files at run time, it will speed up the search process.

Conclusion

Cloud computing attracts, inter alia, with scalability and cost reduction. However, cloud's benefits are accompanied by potential harm to the users data. Existing cloud solutions can be considered synonymous with unknown locations and potentially hostile environments. Therefore security protection objectives can not be guaranteed in the cloud. Motivated by cloudâĂŹs potential we proposed a way to get rid of this drawback. We presented π-Cloud, a personal secure cloud, that enables users to benefit from cloud computing and retain data sovereignty by federating the users own resources. More precisely we presented the prototypic implementation of a Resource Manager for personal secure clouds. This includes the coordination of devices which form this cloud as well as the description of the services they provide and the description of external services. For this description an intermediary format, the Cloud Resource Description Language (CRDL) was developed as an extension of the Open Cloud Computing Infrastructure (OCCI). Since OCCI is currently limited to infrastructure services the approach has been extended to support PaaS and SaaS services as well. Furthermore, an architecture for the coordination and management of CRDL service descriptions and the respective services was developed. As a foundation for the π-Box the Resource Manager enables users to (re)gain their data sovereignty when going to the cloud. Scalability and usability of our prototype have been empirically demonstrated by extensive lab tests.

Future work

Until now we have neglected the need for a trust management system although it represents an inevitable requirement for real life scenarios. However, the estimation of trust in services, resources and provider implies manifold research challenges from different scientific disciplines. Thus, we have postponed this issue for future work.

More technical aspects for further development include distribution support, temporary fragmentation or interconnecting several π-Clouds in order to form a community cloud. As each component of the π-Box can be considered to be a service, it should be possible to distribute them throughout the π-Cloud. This implies mechanisms to handle abrupt disconnections of devices which host services, intelligent replication of services, service redundancy and so on. The scientific area of peer-to-peer protocols offers a variety of potentially suitable basic technologies for this challenge.

For scenarios like business trips it seems plausible that a user would benefit from a mobile π-Box. Building up on already implemented basic π-Box status functionalities, intelligent hand over mechanisms can be developed. Depending on the amount of data necessary for a business trip, it might be wise to migrate the π-Box and relevant data in advance. Particularly interesting is the question which criteria can be used to predict such user behaviour and how this data can be aggregated, utilised and secured.

Finally, in community cloud scenarios every participating π-Box would provide at least partial access to its resources for friendly π-Boxes in addition to its own devices. This entails authentication checks during search, update, delete and retrieval processes.

Competing interests

The authors declare that they have no competing interests.

Author's contributions

All the listed authors made substantive intellectual contributions to the research and manuscript. Specific details are as follows: MM: Led the work on requirements gathering and analysis, design, development and evaluation of the Resource Manager. Carried out the analysis of related work. Wrote the main part of the paper based on his PhD thesis. Contributed to the editing and preparation of the paper as well as the overall architecture of the π-Box. SG: FlexCloud project lead. Responsible for the overall technical approach and architecture, editing and preparation of the paper. Contributed to requirements gathering for the Resource Manager. AS: Principal investigator of the FlexCloud project. All authors read and approved the final manuscript.

Acknowledgements
This work has received funding under project number 080949277 by means of the European Regional Development Fund (ERDF), the European Social Fund (ESF) and the German Free State of Saxony. The information in this document is provided as is, and no guarantee or warranty is given that the information is for any particular purpose. The authors would like to acknowledge the contributions of Hartmut Schweizer in the initial version of this paper [8] and his support in the design and development of CRDL. Furthermore, we appreciate the assistance of Ievgenii Maksymiuk in the development and evaluation of the Resource Manager prototype.

Received: 2 December 2013 Accepted: 4 June 2014
Published online: 26 June 2014

References

1. Mell P, Grance T (2011) The NIST definition of cloud computing. Technical Report 800-145. National Institute of Standards and Technology (NIST), Gaithersburg, MD, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
2. Rochwerger B, Breitgand D, Eliezer L, Galis A Nagin K, Llorente IM, Montero R, Wolfsthal Y, Elmroth E, Cáceres J, Emmerich W, Galán F (2009) The reservoir model and architecture for open federated cloud computing. IBM J Res Dev 53(4):4:1–4:11
3. Flexible Service Architectures for Cloud Computing, FlexCloud project web site. http://flexcloud.eu/. Accessed 2014-06-12
4. Strunk A, Mosch M, Groß S, Thoß Y, Schill A (2012) Building a flexible service architecture for user controlled hybrid clouds In: 7th International Conference on Availability, Reliability and Security, AReS 2012. IEEE, Prague, CZ
5. Groß S, Schill A (2012) Towards user centric data governance and control in the cloud. In: Camenisch J, Kesdogan D (eds) Open Problems in Network Security. Lecture Notes in Computer Science, vol 7039. Springer, Berlin/Heidelberg, pp 132–144
6. Moltkau B, Thoß Y, Schill A (2013) Managing the cloud service lifecycle from the user's view. In: Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER). SCITEPRESS Digital Library, Aachen
7. Spillner J, Schad J, Zepezauer S (2013) Personal and federated cloud management cockpit. PIK – Praxis der Informationsverarbeitung und Kommunikation 36(1):44
8. Mosch M, Schweizer H, Groß S, Schill A (2012) Automated federation of distributed resources into user-controlled cloud environments. In: 5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2012), 2nd International Workshop on Intelligent Techniques and Architectures for Autonomic Clouds (ITAAC 2012). IEEE, Chicago, Il, USA, pp 321–326
9. Akitio My Personal Cloud Server Product web site. http://www.akitio.com/information-center/personal-cloud-server. Accessed 2014-06-12
10. mydlink^{TM}. Product web site. http://www.dlink.com/de/de/home-solutions/mydlink/what-is-mydlink. Accessed 2014-06-12
11. Synology Cloud Services. Product web site. http://www.synology.com/dsm/home_file_sharing_cloud_station.php. Accessed 2014-06-12
12. LenovoEMC Personal Cloud. Product web site. http://www.iomegacloud.com/landing_page.php. Accessed 2014-06-12
13. Samsung Tomorrow. Samsung Electronics Official Global Blog. Samsung HomeSync creates connected media experience for the whole family. http://global.samsungtomorrow.com/?p=22348. Accessed 2014-06-12
14. myQNAPcloud Product web site. https://www.myqnapcloud.com/. Accessed 2014-06-12
15. ownCloud. Project web site. http://owncloud.org/. Accessed 2014-06-12
16. Riva O, Yin Q, Juric D, Ucan E, Roscoe T (2011) Policy expressivity in the anzere personal cloud. In: Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11). ACM, New York, pp 1–14
17. Ardissono L, Goy A, Petrone G, Segnan M (2009) From service clouds to user-centric personal clouds. In: Cloud Computing (CLOUD-II), 2009 IEEE International Conference On. IEEE, Bangalore, India, pp 1–8
18. Cunsolo VD, Distefano S, Puliafito A, Scarpa M (2009) Volunteer computing and desktop cloud: The cloud@home paradigm. In: Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium On. IEEE, Cambridge, MA, USA, pp 134–139
19. Andrzejak A, Kondo D, Anderson DP (2010) Exploiting non-dedicated resources for cloud computing. In: Network Operations and Management Symposium (NOMS), 2010 IEEE. IEEE, Osaka, Japan, pp 341–348
20. Marinos A, Briscoe G (2009) Community cloud computing. In: Proceedings of the 1st International Conference on Cloud Computing (CloudCom'09). Springer, Berlin, Heidelberg, pp 472–484
21. Barraca JP, Matos A, Aguiar RL (2011) User centric community clouds. Wireless Personal Commun 58(1):31–48
22. Ning W, De X, Baomin X (2010) Collaborative integration and management of community information in the cloud. In: E-Business and E-Government (ICEE), 2010 International Conference On. IEEE, Guangzhou, China, pp 1406–1409
23. Chard K, Caton S, Rana O, Bubendorfer K (2010) Social cloud: Cloud computing in social networks. In: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference On. IEEE, Miami, Florida, USA, pp 99–106
24. Kannan S, Gavrilovska A, Schwan K (2011) Cloud4home – enhancing data services with @home clouds. In: Distributed Computing Systems (ICDCS), 2011 31st International Conference On. IEEE, Minneapolis, Minnesota, USA, pp 539–548
25. NVIDIA CUDA Parallel Programming and Computing Platform. http://www.nvidia.com/object/cuda_home_new.html. Accessed 2014-06-12
26. Khronos Group. OpenCL - The open standard for parallel programming of heterogeneous systems. http://www.khronos.org/opencl/. Accessed 2014-06-12
27. Amazon Web Services Homepage. Announcing Cluster GPU Instances for Amazon EC2. http://aws.amazon.com/about-aws/whats-new/2010/11/15/announcing-cluster-gpu-instances-for-amazon-ec2/. Accessed 2014-06-12
28. Brutlag JD, Hutchinson H, Stone M (2008) User preference and search engine latency. In: JSM Proceedings, Qualitiy and Productivity Research Section. IEEE, Alexandria, VA, USA
29. IETF Zeroconf Working Group homepage. http://www.zeroconf.org/. Accessed 2014-06-12
30. Apple Bonjour support homepage. http://www.apple.com/support/bonjour/. Accessed 2014-06-12
31. Avahi project web site. http://www.avahi.org/. Accessed 2014-05-10
32. Open Cloud Computing Interface. Project web site. http://www.occi-wg.org/. Accessed 2014-06-12
33. Open Grid Forum. Project web site. http://www.ogf.org/. Accessed 2014-06-12
34. Metsch T, Edmonds A, Nyrén R (2010) Open cloud computing Interface-Core. In: Open Grid Forum, OCCI-WG, Specification Document. http://forge.gridforum.org/sf/go/doc16161, Online publication
35. Metsch T, Edmonds A (2010) Open cloud computing Interface–Infrastructure. http://ogfweb.pti.iu.edu/Public_Comment_Docs/Documents/2010-12/ogf_draft_occi_infrastructure.pdf, Online publication, Accessed 2012-11-08
36. Open Cloud Computing Interface. OCCI implementations. http://www.occi-wg.org/community/implementations/. Accessed 2014-06-12
37. HyperSQL database. Project web site. http://hsqldb.org. Accessed 2014-06-12
38. Apache Lucene. Project web site. http://lucene.apache.org/core/. Accessed 2014-06-12
39. elasticsearch. Project web site. http://www.elasticsearch.org/. Accessed 2014-06-12

doi:10.1186/s13677-014-0010-8
Cite this article as: Mosch *et al.*: User-controlled resource management in federated clouds. *Journal of Cloud Computing: Advances, Systems and Applications* 2014 **3**:10.