Scalable Computing: Practice and Experience

Scientific International Journal for Parallel and Distributed Computing

ISSN: 1895-1767

Universitatea de Vest din Timişoara

Volume 14(4) December 2013

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 14, Number 4, December 2013

TABLE OF CONTENTS

DOI 10.12694/scpe.v14i4.926
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. iii–iv. http://www.scpe.org

INTRODUCTION TO THE SPECIAL ISSUE ON CLOUD FOR INTERNET OF THINGS

Dear SCPE readers,

The Internet of Things (IoT) changes the way we interact with the world around us. It aims to represent the physical world through uniquely identifiable and interconnected objects (things). Things have the capacity for sensing, processing or actuating information about entities available from within the real world. Thus, information travels along heterogeneous systems, such as routers, databases, information systems and the Internet, leading in the generation and movement of enormous amounts of data which have to be stored, processed and presented in a seamless, efficient and easily interpretable form.

Cloud computing represents a very flexible technology, able to offer theoretically unlimited computing and storage capabilities, and efficient communication services for transferring terabyte flows between data centres. Both the IoT and Cloud technologies address two important goals for distributed system: high scalability and high availability. All these features make the Cloud Computing a promising choice for supporting IoT services. IoT can appear as a natural extension of Cloud Computing implementations, where the Cloud provides IoT based resources and capabilities, process IoT data, manage IoT environments and deliver on-demand utility for IoT services, such as sensing/actuation as a service.

This special issue aims to gather innovative works on Cloud solutions for integrating monitors and sensors, storage devices, analytics, tools and virtualization platforms, in order to support IoT purposes. It includes extended, thoroughly revised papers presented at the Workshop on CLoud for IoT (CLIoT 2013) and Workshop on CLOUd Storage Optimization (CLOUSO 2013), which have been organized in conjunction with the European Conference on Service-Oriented and Cloud Computing (ESOCC 2013), in Malaga, Spain, on September 11th, 2013.

The papers included in this special issue deal with several issues, which aim to support efficient IoT applications and services. Tomarchio et al. [1] present an OSGi-based middleware, called Sensor Node Plug-in System (SNPS), able to abstract sensors from their proprietary interfaces and to offer their capabilities to third party applications according to an as-a-Service approach. Thus, sensors are no longer low-level devices producing raw measurement data, but can be seen as services to be used and composed over the Internet in a simple and standardized way.

Fazio et al. [2] discuss the design of a Message Oriented Middleware for Cloud, called MOM4C, able to arrange customizable Cloud facilities by means of a flexible federation-enabled communication system. They focus their discussion on the Dangerous Good Transportation (DGT) use case in order to show the applicability of the proposed middleware in IoT scenarios. To this aim, specific MOM4C utilities are combined in order to compose a PaaS for sensed data drawing, storage and processing.

Destefano et al. [3] present the Sensing and Actuation as a Service (SAaaS) architecture for enrolling and aggregating sensing resources into the Cloud. Specifically, they investigate a sensing resource abstraction solution designed for mobile devices, called SAaaS4Mobile. The implementation of SAaaS4Mobile abstraction modules has been tackled for Android mobiles and are based on well-known standards, as the Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE).

Bellavista et al. [4] provide to both IaaS Cloud providers and to SaaS application providers an open source tool that facilitates the composition of heterogeneous resources, such as single Virtual Machines (VMs), DB services and storage, and stand-alone services, in order to make smart objects for the IoT. The tool automates the provisioning of complex SaaS applications over the widely diffused real-world open-source OpenStack IaaS, integrating well-known technologies, such as the standard Business Process Execution Language (BPEL), which simplifies the definition of the deployment, configuration, and monitoring steps.

Veltri et al. [5] propose a constrained version of the Session Initiation Protocol (SIP), named CoSIP, which allows constrained devices to instantiate communication sessions in a lightweight and standard fashion and can be adopted in M2M application scenarios. The proposed CoSIP is a binary protocol which maps to SIP, similarly to CoAP doesto HTTP. CoSIP can be adopted in several application scenarios, such as service discovery and publish/subscribe applications.

To support storage services for IoT, Villari et al. [6] introduce an abstraction layer that works above heterogeneous Cloud storage platforms, able to split data in many chunks spread over different storage providers.

Users do not need to take care about a specific provider for data upload /download and experience a seamless storage service, where storage space is almost the sum of the storage spaces offered by the involved Cloud providers. At the same time, Data Obfuscation is guaranteed. In fact, Cloud providers can not have full access to the stored files, since they store few chunks.

In the context of the VISION Cloud EU-funded project, which aims to design a new scalable and flexible storage Cloud architecture, Buneo et al. [7] investigate a storage Cloud environment, where a distributed file system is built on top of a set of storage nodes composing a cluster, and several clusters constitute a Cloud data center. The authors provide an analytic model based on Stochastic Reward Nets (SRNs) to analyze the reached availability level of a VISION Cloud storage cluster varying both structural and timing system parameters.

With reference to the VISION Cloud project, Villari et al. [8] describe a delegation architecture for on-boarding federation, which allows an enterprise to efficiently migrate data from one storage Cloud provider to another, while providing continuous access and a unified view over the data during the migration. They provide implementation details based on Security Assertion Markup Language (SAML), a protocol designed for delegation issues with strong security features.

REFERENCES

[1] Orazio Tomarchio, Giuseppe Di Modica, Francesco Pantano, *An OSGi Middleware to Enable the Sensor as a Service Paradigm.*

[2] Maria Fazio, Antonio Celesti, Antonio Puliafito, Massimo Villari, *A message oriented middleware for Cloud computing to improve efficiency in risk management systems.*

[3] Salvatore Distefano, Giovanni Merlino, Antonio Puliafito, *Mobiles for Sensing Clouds: the SAaaS4Mobile experience.*

[4] Paolo Bellavista, Antonio Corradi, Luca Foschini, Alessandro Pernafini, *Towards an Automated BPEL-based SaaS Provisioning Support for OpenStack IaaS.*

[5] Luca Veltri, Simone Cirani, Marco Picone, *A Session Initiation Protocol for the Internet of Things.*

[6] Massimo Villari, Antonio Celesti, Maria Fazio, Antonio Puliafito, *Evaluating a File Fragmentation System for Multi-cloud Storage Services.*

[7] Dario Bruneo, Francesco Longo, David Hadas, Hillel Kolodner, *Analytical Investigation of Availability in a Vision Cloud Storage Cluster.*

[8] Ciro Formisano, Elliot K. Kolodner, Alexandra Shulman-Peleg, Ermanno Travaglino, Gil Vernik and Massimo Villari, *Delegation Across Storage Clouds: On-boarding Federation as a Case Study.*

Maria Fazio, *DICIEAMA Department, University of Messina, Italy* (`mfazio@unime.it`)
Nik Bessis, *School of Computing and Mathematics, University of Derby, UK* (`N.Bessis@derby.ac.uk`)

DOI 10.12694/scpe.v14i4.927
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. 188–200. http://www.scpe.org

AN OSGI MIDDLEWARE TO ENABLE THE SENSOR AS A SERVICE PARADIGM

GIUSEPPE DI MODICA, FRANCESCO PANTANO, ORAZIO TOMARCHIO *

Abstract. The Internet of Things vision has recently stimulated many research efforts in different communities. The collection of diverse information produced by the proliferation of inter-connected sensing entities is one of the biggest challenge that should be adequately addressed. The huge amounts of raw data produced by IoT devices need to be structured, stored, analysed, correlated and mined in a reliable and scalable way. The size of the produced data, and the high rate at which data are being produced, suggest that we need new solutions that combine tools for data management and services capable of promptly structuring, aggregating and mining data even just at the time they are produced. In this paper we propose a middleware, to be deployed on top of physical sensors and sensor networks, capable of abstracting sensor devices from their proprietary interfaces, and offering them to third party applications in an as-a-Service fashion for prompt and universal use. The middleware also offers tool to elaborate real-time measurements produced by sensors. A prototype of the middleware has been implemented. In the paper a real use case scenario is also discussed.

1. Introduction. The current role of the Internet of Thing (IoT) is no more limited only to electronic identification of objects but it is perceived as a way to act, measure, or provide services based on real-world entities [20]. Advancements in networking technologies and sensor/actuator capabilities provide a large number of physical world objects with communication and computation capabilities to interact with their surrounding environment.

However, the emerging IoT platforms are currently hard to deploy and operate, requiring except in the most trivial cases the intervention of domain experts to interpret the sensor data and, eventually, to come up with actuation commands. This approach is clearly too expensive and time-consuming, and simply does not scale with the increasing number of IoT devices. In addition, the development of IoT application is highly scenario and technology dependent, due to the heterogeneity of devices. That is why powerful middleware solutions are required to integrate heterogeneous devices and dynamic services for building complex systems for the IoT.

We believe that the service-oriented approach [14, 24] provides adequate abstractions for application developers, and that it is a good approach to integrate heterogeneous sensors and different sensor network technologies with Cloud platforms through the Internet, by paving the way for new IoT applications.

In this paper, extending the work presented in [8] we propose an OSGi-based middleware, called *Sensor Node Plug-in System (SNPS)*, where sensors are no longer low-level devices producing raw measurement data, but can be seen as "services" to be used and composed over the Internet in a simple and standardized way in order to build even complex and sophisticated applications.

The remainder of the paper is structured in the following way. Section 2 presents a review of the literature. Sect. 3 introduces the architecture of the proposed solution. In Sect. 4 we discuss and motivate the choice of the data model implemented in the middleware. Section 5 provides some details on the sensor composition process. In Sect. 6 a use case scenario is discussed. We conclude our work in Sect. 7.

2. Related work. The most notably effort in providing standard definition of Web service interfaces and data encodings to make sensors discoverable and accessible on the Web is the work done by the Open Geospatial Consortium (OGC) within the Sensor Web Enablement (SWE) initiative [4, 23]. The role of the SWE group is to develop common standards to determine sensors capabilities, to discover sensor systems, and to access sensors' observations. The principal services offered by SWE include:

- Sensor Model Language (SensorML): provides a high level description of sensors and observation processes using an XML schema methodology
- Sensor Observation Service (SOS): used to retrieve sensors data.
- Sensor Planning Service (SPS): used to determine if an observation request can be achieved, determine the status of an existing request, cancel a previous request, and obtain information about other OGC web services
- Web Processing Service (WPS): used to perform a calculation on sensor data.

*Department of Electrical, Electronic and Computer Engineering, University of Catania, Catania, Italy (firstname.lastname@dieei.unict.it).

A common misconception of the adoption of SWE standards is that, rather than encapsulating sensor information at the application level, they were originally designed to operate directly at the hardware level. Of course, supporting interoperable access at the hardware level has some advantages as it copes well with the "plug and play" concept. Currently, some sensor systems such as weather stations and observation cameras already offer access to data resources through integrated web servers. However, besides contradicting the view of OGC's SWE of uncoupling sensor information from sensor systems, the downside of this approach becomes clearer when dealing with a high number of specialized and heterogeneous sensor systems, and in particular in resource-limited scenario where communication and data transportation operations must be highly optimized. Even a relatively powerful sensor gateway has not the basic requirement to act as a web server in many cases, it is networked via a low-bandwidth network and is powered by a battery, so it has neither the energy nor bandwidth resources required to run a web service interface.

The need for an intermediate software layer (middleware) derives from the gap between the high-level requirements of pervasive computing applications and the complexity of the operations involved in the underlying Wireless Sensor Networks (WSNs). A WSN is characterized by constrained resources, a dynamic network topology, and low level embedded OS APIs, while the application requirements include high flexibility, re-usability, and reliability to cite a few. In general, WSN middlewares help the programmer develop applications by providing appropriate system abstractions, reusable code services and data services, flexible network infrastructure management and efficient resource management services.

Some research efforts have surveyed the different approaches of middleware and programming paradigms for WSN: in [11] middleware challenges and approaches for WSN were analysed, while [26] and [22] analysed programming models for sensor networks. In the following we focus on two programming models which in the past years have been taken in great consideration by researchers: the *message-oriented* and the *service-oriented* approach.

Message oriented middlewares. Message oriented middlewares (MOMs) provide distributed communication among participants on the basis of the *asynchronous* interaction model. Basically, in a MOM a participant (sender) who need to communicate to another participant (receiver) produces a message which is managed by a software queue which, in its turn, takes care of routing that message to the final destination. Two types of interaction models can be implemented in a MOM: point-to-point, by which a single message produced by a sender is delivered only once to only one receiver, and publish/subscribe, which allows a single producer to send a message to or potentially hundreds of thousands of receivers.

Many sensor frameworks calls on MOM as asynchronous communication can guarantee very high energy saving in event-driven WSN applications. Most of them opted for the publish/subscribe mechanism to implement message exchange between sensors and nodes. Mate [17] uses a Virtual Machine approach as level of abstraction. It deals with several issues such as limited bandwidth and energy consumption for large networks. Claimed features are fault tolerance, dynamism, flexibility and reconfigurability components. The weak point of this middleware is, however, energy expenditure, mainly due to generated communication overhead; therefore is not suitable for complex applications. Magnet [3] is a Virtual Machine based middleware as well. It works as an abstraction layer built on top of MagneOS, an adaptive operative system sensor oriented that allows to detect and report any moving object. The weak point is that the entire network is based on a single Java Virtual Machine running static components, so the performance is acceptable only under very stringent constraints. Impala [19] is an event-driven middleware which manages requests and responses in a completely asynchronous mode. It adapt itself to many application scenarios and can automatically set parameters according to the presented scenarios. It introduces a small overhead in transmission phase. Milan [12] allows to integrate applications on different networks. It lets applications specify QoS requirements and adjust network characteristics to improve the life cycle management. These goals are achieved by collection data from sensors aggregated on the network, with particular care for energy consumption and bandwidth usage. Cougar [7] is a database oriented middleware that stores sensor and data in a relational fashion. The management operations of WSNs are implemented in the form of SQL queries. Sina [25] presents an architecture which adapts to the physical environment. In fact, the network is modelled as a distributed system of objects, queried according the SQL language. Through the hierarchical clustering mechanism, Sina caters for scalability, as the routing of requests takes place in a hierarchical manner as a function of the root node. DsWare [18] is a middleware that uses a database approach

and is also network event-driven. Its architecture is based on modules that deal with the management of data, group management, detection of events. The publish/subscribe mechanism is used for asynchronous, topic-based communication. As Cougar, DsWare uses SQL for the registration and cancellation of events.

Service oriented middlewares. In a service oriented approach participants of a distributed system communicate through the *synchronous* interaction model. Service, rather than message, is the focus of service oriented middlewares (SOMs). This approach greatly facilitates the design task (service-centric): adding advanced features simply means implementing new services. If compared to the MOM, SOMs propose a more straightforward approach to messaging, but suffer from inflexibility and tight coupling (potential geometric growth of interfaces) between the communicating participants.

Recently, the service-oriented approach has been applied to sensor environments [14, 21]. The common idea of these approaches is that, in a sensor application, there are several common functionalities that are generally irrelevant to the main application. For example, most services will have to support service registries and discovery mechanisms and they will also need to provide some level of abstraction to hide the underlying environments and implementation details. Furthermore, all applications need to support some levels of reliability, performance, security, and QoS. All of these can be supported and made available through a common middleware platform instead of having to incorporate them into each and every service and application developed.

The SStreaMWare [10] middleware exploits a query-based system in order to access the data collected. Its service oriented nature solves the problem of software heterogeneity, facilitating the integration and guaranteeing compatibility on the one end, and allowing a weak coupling between the user application and the sensors on the other end. Useme [5] provides a high-level programming language to develop applications for wireless sensor network. The service oriented approach ensures scalability, interoperability and efficiency. Oasis [16] is an ambient-aware, service oriented programming model. The object-oriented paradigm provides an abstraction that focuses on the monitored physical phenomenon, bypassing the complexity of the network topology. MiSense [15] is a service-oriented middleware that provides a publish-subscribe mechanism based on well-defined contents. It is able to reduce the complexity through a structure above the component layers by imposing restrictions on the modularity and offering a well defined and specific interface to the rest of the system. UbiSOAP [6] is designed on the SOAP protocol. Its aim is to provide services on a pervasive sensors network. It's composed two layers, a multi-radio network layer and a communication-oriented Web Services layer. A SOAP Transport service is used by the client and the service: the Transport service interacts with the multi-radio network, sending and receiving messages over the network. TinySOA [2] is a service-oriented architecture that provides developers with simple API to implement applications on sensor networks, using the same programming language in which they were originally written. The main advantage of TinySOA is to abstract away the developer from details of the WSN hardware and the communication layer.

The service oriented approach is the one we adopt in the work being presented. In particular, we leverage on OSGi [1] as the key paradigm enabling the *service* technology in the context of sensor networks. Furthermore, the choice of the service oriented paradigm allows for a more smooth integration with Cloud platforms and for advanced discovery mechanisms also employing semantic technologies [9]. The work we propose aims at reaching a worthy compromise between WSN heterogeneity, system scalability and interoperability.

The middleware we propose on the one hand embeds service oriented features in that functionalities are provided end exposed through services; on the other one the message oriented paradigm is used to manage events produced by sensors. This way advantages of both approaches are then caught. Further, the sensor aggregation feature offered by the middleware contributes to enhances the flexibility of the sensor programming model. By means of this novel service, third party applications are exposed a new way to design and implement data manipulation schemes which relieves them from the burden of gathering, putting together and elaborate on all data coming from their sensing campaigns.

3. The SNPS middleware. This section presents the proposal for a middleware devised to lay on the physical layer of wireless sensors, abstract away the sensors' specific features, and turn sensors into smart and composable services accessible through the Internet in an easy and standardized way. The middleware was designed to follow the basic principles of the IoT paradigm [20]. Sensors are not just sources of raw data, but are seen like smart objects capable of providing services like filtering, combining, manipulating and delivering information that can be readily consumed by any other entity over the Internet according to well-known and

standardized techniques.

Primary goal of the middleware, which we called *Sensor Node Plug-in System (SNPS)*, is to bring any physical sensor/actuator on an abstraction level that allows for easier and standardized management tasks (switch on/off, sampling), in a way that is independent of the proprietary sensor's specification. By the time a sensor is "plugged" into the middleware, it will constitute a resource/service capable of interacting with other resources (be them other sensors plugged into the middleware or third party services) in order to compose high-value services to be accessed in SOA-fashion. The middleware also offers a set of complimentary services and tools to support the management of the entire life cycle of sensors and to sustain the overall QoS provided by them.

Basically, the SNPS can be said to belong to the category of the service-oriented middlewares [24]. In fact, the provided functionality are exposed through a service-oriented interface which grants for universal access and high interoperability. Yet, all data and information gathered by sensors are stored in a database that is made publicly accessible and can be queried by third party applications. Further, the SNPS also support asynchronous communication by implementing the exchange of messages among entities (sensors, components, triggers, external services). All these features makes the middleware flexible to any application's need in any execution environment.

At design time it was decided not to implement the entire middleware from scratch. A scouting was carried out in order to identify the software framework that best supported, in a native way, all the characteristics of flexibility and modularity required by the project. Eventually, the OSGi framework [1] was chosen. The OSGi framework natively supports the component's life cycle management, the distribution of components over remote locations, the seamless management of components' inter-dependencies, and an asynchronous (event-based) communication paradigm.

The SNPS middleware is then organized into several components, each of which is implemented as a software module (or "bundle") within the OSGi framework. Figure 3.1 shows the architecture of the middleware and its main components.

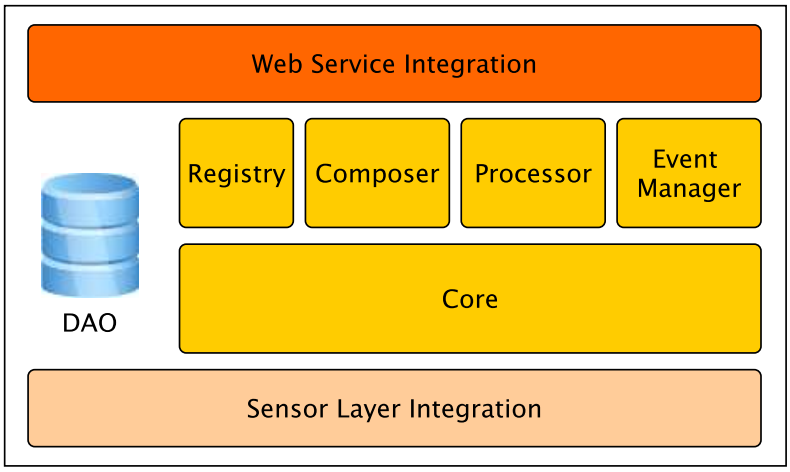


Fig. 3.1. *SNPS architecture*

The overall architecture can be broken down into three macro-blocks:

- Sensor Layer Integration
- Core and related Components
- Web Service Integration

In the following we provide a description of each macro-block.

3.1. Core and related Components. The components we are about to discuss are charged the responsibility of providing most of the middleware's functionality. In Figure 3.2 the connections among the components

are shown.

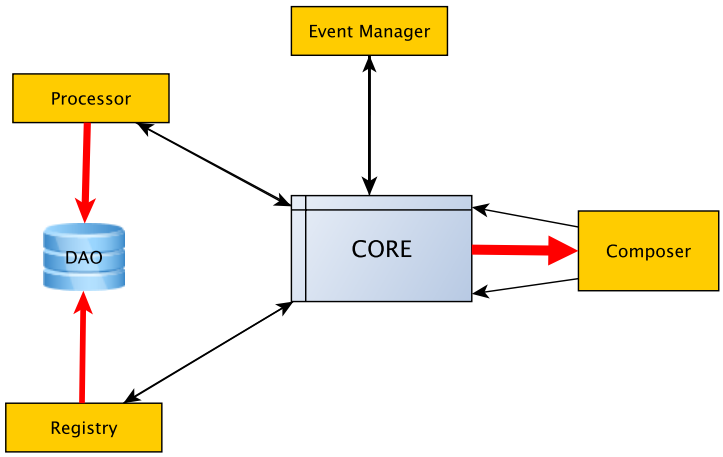


FIG. 3.2. *Core and related Components*

Core. It is where the business logic of the Middleware resides. The Core acts as an orchestrator that coordinates the middleware's activities. Data and commands flowing forth and back from the web service layer to the sensor layer are dispatched by the Core to the appropriate component. This component is responsible for the virtualization of physical sensors, i.e., conforming the features and capabilities of physical sensors under a common representation. In addition, the core is able to define the working policies of the middleware, identifying points of failure and performing action in order to recover components.

Registry. It is the component where all information about sensors, middleware's components and provided services are stored and indexed for search purpose. As for the sensors, data regarding the geographic position and the topology of the managed wireless sensor networks are stored in the Registry. Also, each working component needs to signal its presence and functionality to the Registry, which will have to make this information public and available so that it can be discovered by any other component/service in the middleware.

Composer. It represents the component which implements the sensors' composition service. Virtualized sensors have a uniform representation which allows for "aggregating" multiple sensors into just one sensor that will eventually be exposed to applications. An insight and practical examples about the aggregation service is provided in Section 5.

Processor. It is the component responsible for the manipulation of the data coming from the sensors. It enforces the sensors' aggregation schemes defined through the Composer. In particular, when data come from multiple sensors composing an aggregate, this component applies the directives of the related aggregation mechanism.

Event Manager. It is one of the most important components of the middleware. It provides a publish/-subscribe mechanism which can be exploited by every middleware's component to implement asynchronous communication. Components can either be producers (publishers) or consumers (subscribers) of every kind of information that is managed by the middleware. This way, data flows, alerts, commands are wrapped into "events" that are organized into topics and are dispatched to any entity which has expressed interest in them.

DAO. It represents the persistence layer of the middleware. It exposes APIs that allow service requests to be easily mapped onto storage or search calls to the database.

3.2. Sensor Layer Integration. The Sensor Layer Integration (SLI) represents the gateway connecting the middleware to the physical sensors. It implements a *bidirectional communication channel* supporting commands to flow both from the middleware to the sensors and from the sensors to the middleware as well and a *data channel* (for data that are sampled by sensors and need to go up to the middleware).

The addressed scenario is that of wireless sensor networks implemented through so called Base Stations (BS) to which multiple sensors are "attached". A BS implements the logic for locally managing its attached sensors. Sensors can be wiredly or wirelessly attached to a BS, forming a network which is managed according to specific communication protocols, which are out of our scope. The SLI will then interact just with the BS, which will only expose its attached sensors hiding away the issues related to the networking.

The integration is realized by means of two symmetrical bundles, which are named respectively *Middleware Gateway bundle (iMdmBundle)* and *WSN Gateway Bundle (iWsnBundle)*. The former lives in the middleware's runtime context, and was thought to behave as a gateway for both commands and data coming from the BSs and directed to the middleware; the latter lives (runs) in the BS's runtime context, and forwards commands generated by the middleware to the BSs. Since the middleware and the BSs may be attached to different physical networks, the communication between the two bundles is implemented through R-OSGi, which is a specific OSGi's bundle offering a remote communication service which other bundles living in different runtime contexts can use to communicate to each other. In Figure 3.3 the two bundles and their respective runtime contexts are shown.

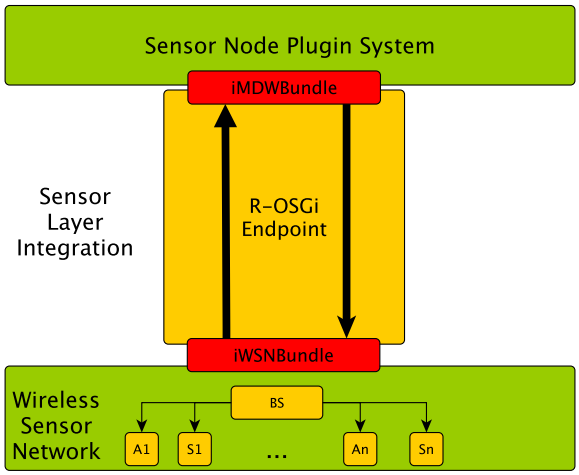


FIG. 3.3. *OSGi bundles implementing the Sensor Layer Integration*

The SLI was designed to work with any kind of BS, independently of the peculiarity of the sensors it manages, with the aim of abstracting and making uniform the access to sensors' functionality. Making uniform the management of the sensors' life cycle does not mean giving up the specific capabilities of sensors. Physical sensors will maintain the way they work and their peculiar features (in terms, for instance, of maximum sampling rate, sampling precision, etc.). But, in order for sensors (read base stations) to be pluggable into the middleware and be compliant to its management logic, a minimal set of requirements must be satisfied: the *iWsnBundle* to be deployed on the specific BS will have to interface to the local BS' logic and implement the functionality imposed by the SNPS middleware (switch on/off sensors, sample data, run sampling plan) by invoking the proprietary base station's API.

3.3. Web Service Integration. As it is shown in Fig. 3.4, the *OSGi bundle Wrapper* exports the functionality of the SNPS middleware to a Web Service context.

A modular system bundle-based enables applications that run on different platforms to communicate with each other.

This result has been achieved by exploiting the OSGi communication model; different can communicate not only importing and exporting services, but using SOA strategies Web services based.

In this case, Web services have been built using Apache CXF, which is an open source services framework that helps you build and develop services using front-end programming APIs, like JAX-WS and JAX-RS.

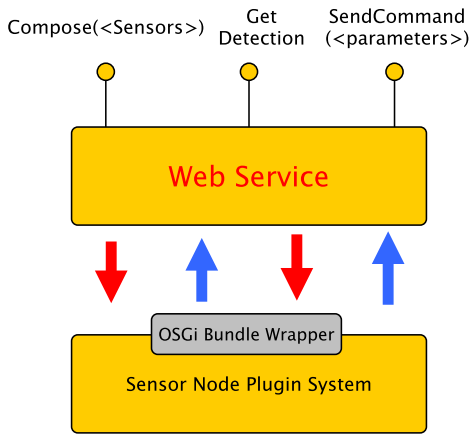


FIG. 3.4. *Wrapping and exposing SPNS as a Web Service*

These services can speak a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP or CORBA, and in this case we select the SOAP protocol, exporting to clients a range of services to access middleware resources. By doing so, you can execute query for a specific sensor and its associated data, according to certain search criteria. In this way clients are able to communicate directly with sensors virtual images by sending commands, including:

- sensor activation
- sensor deactivation
- sending sampling plans
- request for sensor detection
- sensor composition

In particular, the deactivation command does not perform a proper device shutdown, but allows to change it in a power-saving state, in order to save resources that can be used at a later date.

4. SNPS data model. The SNPS data model is one of the most interesting features of the middleware. Goals like integration, scalability, interoperability are the keys that drove the definition of the model at design time. The objective was then to devise a data model to structure both sensors' features (or capabilities) and data produced by sensors. The model had to be rich enough to satisfy the multiple needs of the middleware's business logic, but at the same time had to be light and flexible to serve the objectives of performance and scalability. We surveyed the literature in order to look for any proposal that might fit the middleware's requirement. Specification like SensorML and O&M [4] seam to be broadly accepted and widely employed in many international projects and initiatives. SensorML is an XML-based language which can be used to describe, in a relatively simple manner, sensors capabilities in terms of phenomena they are able to offer and other features of the specific observation they are able to implement. O&M is a specification for describing data produced by sensors, and is XML-based as well. XML-based languages are known to be hard to treat, and in many cases the burden for the management of XML-based data overcomes the advantage of using rigorous and well-structured languages. We therefore opted for a solution that calls on a reduced set of terms of the SensorML specification to describe the sensor capabilities, and makes use of a much lighter JSON [13] format to structure the data produced by sensors. An excerpt of what a description of sensor capabilities look like is shown in Fig. 4.1.

This is the basic information that must be attached to any sensor before it is plugged into the middleware. Among others, it carries data regarding the phenomena being observed, the sampling capabilities, and the absolute geographic position. When the sensor wakes up, it sends this information to the middleware, which will register the sensor to the Registry bundle, and produce its *virtualized image*, i.e., a software alter-ego of the physical sensor which lives inside the middleware run-time. The virtual sensor has a direct connection with the physical sensor. Each interaction involving the virtual sensor will produce effects on the physical sensor too. It

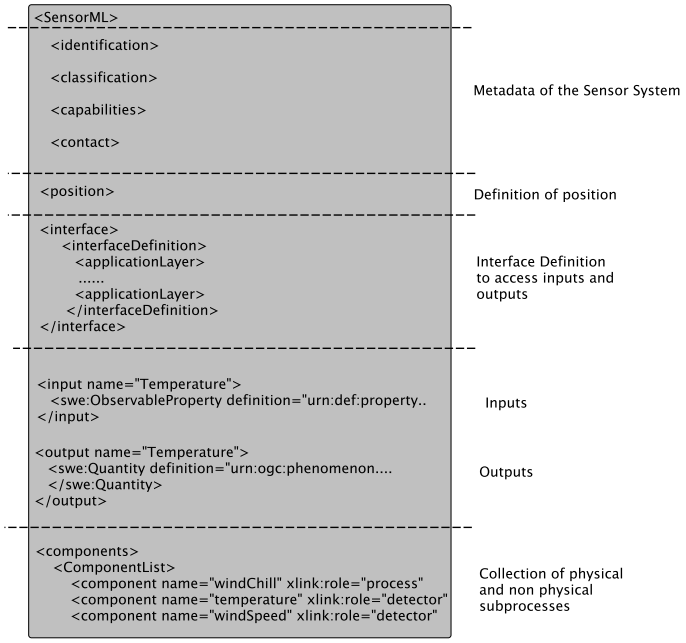


FIG. 4.1. *Description of sensor capabilities in SensorML*

is important to point out that all virtual sensors are treated uniformly by the middleware's business logic.

Furthermore, SensorML is by its nature a process-oriented language. Starting from the atomic process, it is possible to build the so-called *process chain*. We exploited this feature to implement one of the main service provided by the SNPS, i.e., the sensors' composition service (see Sect. 5 for more details). This service, in fact, makes use of this feature to elaborate on measurements gathered by multiple sensors.

As regards the definition of the structure for sensor data, JSON was chosen because it ensures easier and lighter management tasks. The middleware is designed to handle (sample, transfer, store, retrieve) huge amounts of data, with the ambitious goal to also satisfy the requirements of real-time applications. XML-based structures are known to cause overhead in communication, storage and processing tasks, and therefore they do not absolutely fit our purpose. Another strong point of JSON is the ease of writing and analyzing data, which greatly facilitates the developer's task. A data sampled by a sensor will then be put in the following form:

```
Sensor_Measure:
{
    ``SensorId'':``value'',
    ``data'':``value'',
    ``type'':``value'',
    ``timestamp'':``value''
}
```

5. Building and composing Virtual Sensors. Sensor Composition is the most important feature of the SNPS middleware. Simply said, it allows to get complex measurements starting from the samples of individual sensors. This composition service is provided by the Composer bundle (cf. Fig 3.1).

An important prerequisite of the composition is the sensor "virtualization", which is a procedure performed by the Core component when a sensor is plugged into the SNPS middleware (see Sect. 3.2). Aggregates of sensors can be built starting from their software images (virtual sensors) that live inside the SNPS middleware. Therefore, in order to create a new composition (or aggregate) of sensors, the individual virtual sensors to be combined need to be first identified. Secondly, the operation that is to be applied to sensor's measurements

must be specified. This is done by defining the so-called *Operator*, which is a function that defines the expected input and output formats of the operation being performed. The final composition is obtained by just applying the Operator to the earlier chosen virtual sensors.

The operator for aggregating sensors can be defined using MathML [27]. In the operator schema, each sensor input is bound to an item of the formula to be executed. A check is then performed in order to verify the compatibility of the unit of measures of the data that are being aggregated by the operator. This feature is very useful in the case of heterogeneous sensors' composition, while in the case of homogeneous sensors pre-defined patterns are offered.

Once the operator has been created, a new virtual sensor (the aggregate) is available in the system and exposed for use by third party applications. Figure 5.1 shows the structure of an aggregate of sensors.

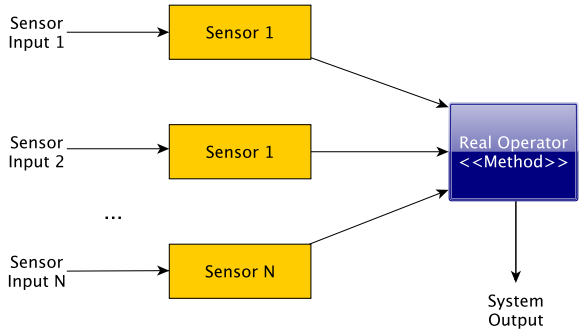


Fig. 5.1. *Sensors composition*

Let us figure out a practical use case of sensor composition. Imagine that there are four temperature sensors available in four different rooms of an apartment. An application would like to know about the instant average temperature of the apartment. A new sensor can be built starting from the four temperature sensors by just applying an *average* operator, as shown in Fig. 5.2.

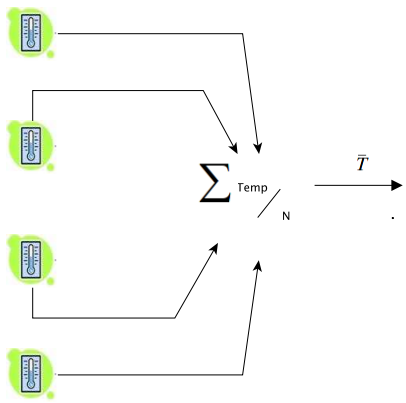


Fig. 5.2. *Average operator*

In this specific case, the input sensors are homogeneous. As stressed earlier, the middleware also provides for the composition of heterogeneous sensors (e.g., temperature, humidity, pressure, proximity), provided that the operator's I/O scheme is adequately designed to be compatible with the sensors' measurement types.

6. Use case scenario. A prototype of the middleware has been implemented and its functionality have been tested. In this section we provide some insight on a real use case that we set up in order to prove the

effectiveness of the implemented mechanisms. In particular, here we focus on what we believe is the most important middleware's provided service, which is the sensors composition service.

The sensors composition process puts emphasis on the semantics of the operation, rather than relying on the simple measure. In this regard, it has been developed a use case, in order to emphasize the power and importance of the aggregation of sensors. Let us recall the average temperature example, and try to describe which are the execution steps triggered in that specific use case.

FIG. 6.1. *Apartment scenario*

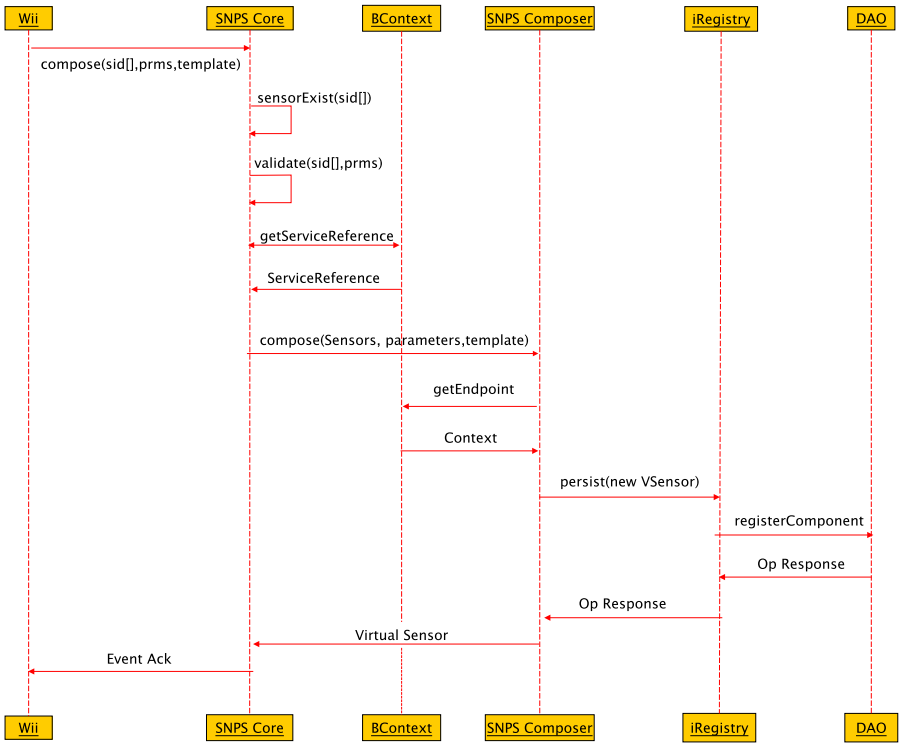


FIG. 6.2. *Sequence diagram for the sensor composition phase*

We will consider the use case as divided into two distinct phases: 1)sensors' composition and 2)aggregate sensor inquiry.

Phase 1: Sensors' composition.. In the first stage, we are going to consider the following actors:

- Web Integration Interface (Wii) Component. It represents the entity generating the composition request;
- Composer. It generates the new virtual sensor from simple temperature sensors;
- Registry. It registers the sensor;
- Core. It orchestrates the composition task among the middleware components.

The operations carried out in the scenario, shown in Fig. 6.2, are the following:

1. The Wii propagates the request to the Core of the platform.
2. The Core retrieves the images of the selected sensors and perform a two-steps validation:
   - Verification of the existence of the sensor images in memory;
   - Validation of the operator to be applied to the sensors;
3. The Core invokes the composition service provided by the Composer;
4. The Composer generates the new (virtual) aggregate sensor;
5. The Registry registers the new sensor;
6. The Core generates a "registration" event for the new sensor, according to the Publish/Subscribe paradigm;

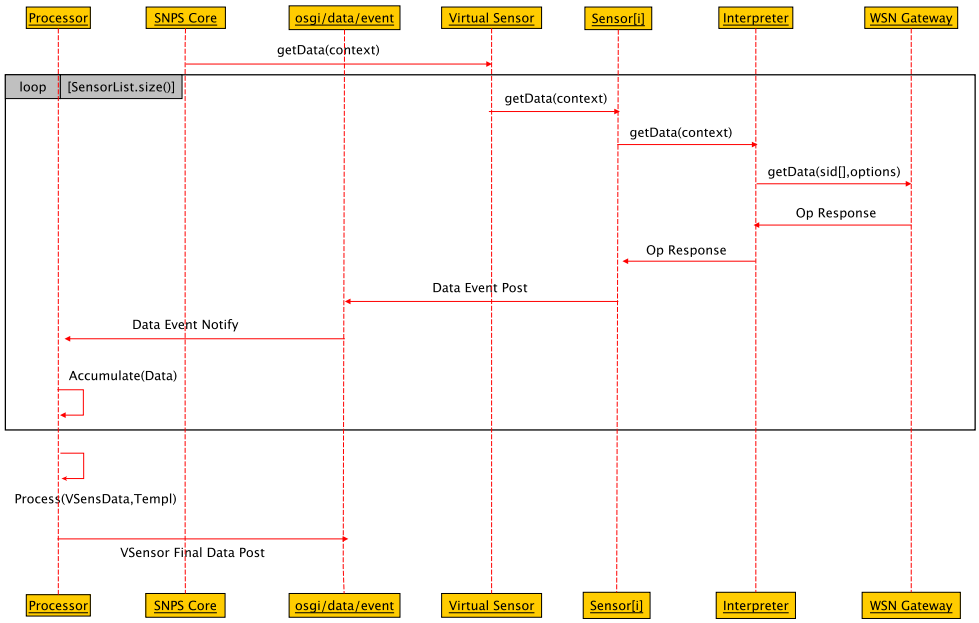


FIG. 6.3. *Sequence diagram for the aggregate sensor inquiry*

Aggregate sensor inquiry.. The steps made in this phase, described in Fig. 6.3, are the following ones:

1. The Wii propagates the request to the Core;
2. The Core, after selecting the aggregate sensor, invokes the get-data operation;
3. The virtual sensor image invokes, for each composing sensor, a service provided by the Sensor Layer Integration (SLI);
4. The SLI propagates the request to the gateway (at WSN Level), which is able to interact directly with the Base Station, which maps the command into a direct command to each physical sensor;
5. After getting the data, the SLI generates a Data response event, which the aggregate sensor is able to collect;
6. The aggregate sensor, finally, applies the operator to the previously collected data, and generates an event on the topic of interest;
8. The Processor records the measurement.

7. Conclusion and future work. The size of data produced by sensors and sensor networks deployed worldwide is growing at a rate that current data analysis tools are not able to follow. Sources of data are multiplying on the Internet (think about smart devices equipped with photo/video cameras). There is a plethora of sensor devices producing information of any kind, at very high rates and according to proprietary specification. This complicates a lot the task of data analysis and manipulation. In this paper we have proposed a solution that aims to ease these tasks. What we propose is not just an early-stage idea but a concrete middleware that implements a mechanism useful to abstract sensors away from their proprietary interfaces and structure, which also offers tool to aggregate and expose sensors and sensor data in the form of services to be accessed in SOA fashion. A prototype of the middleware has been implemented and tested in a small testbed. In the future we are going to conduct extensive experiments to test the scalability and the performance of the middleware in distributed (even geographic) contexts.

Acknowledgment. This work has been partially funded by the Italian project "Sensori" (Industria 2015 - Bando Nuove Tecnologie per il Made in Italy) - Grant agreement n. 00029MI01/2011.

REFERENCES

[1] O. Alliance, *Open Service Gateway initiative (OSGi)*, 2013. Available at http://www.osgi.org/.

[2] E. Aviles-Lopez, and J. Antonio, Garcia-Macias, J. Antonio, *Tinysoa: a service-oriented architecture for wireless sensor networks*, Service Oriented Computing and Applications, (2009).

[3] R. Barr, J. C. Bicket, D. S. Dantas, B. Du, T. W. D. Kim, B. Zhou, and E. G. Sirer, *On the need for system-level support for ad hoc and sensor networks*, SIGOPS Oper. Syst. Rev., (2002), pp. 1–5.

[4] M. Botts, G. Percivall, C. Reed, and J. Davidson, *Ogc sensor web enablement: Overview and high level architecture*, in GeoSensor Networks, S. Nittel, A. Labrinidis, and A. Stefanidis, eds., vol. 4540 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 175–190.

[5] E. Caete, J. Chen, M. Diaz, L. Llopis, and B. Rubio, *Useme: A service-oriented framework for wireless sensor and actor networks*, in Applications and Services in Wireless Networks, 2008. ASWN '08. Eighth International Workshop on, 2008.

[6] M. Caporuscio, P.-G. Raverdy, and V. Issarny, *ubisoap: A service-oriented middleware for ubiquitous networking*, Services Computing, IEEE Transactions on, (2012).

[7] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao, *The cougar project: a work-in-progress report*, SIGMOD Rec., (2003).

[8] G. Di Modica, F. Pantano, and O. Tomarchio, *SNPS: an OSGi-based middleware for Wireless Sensor Networks*, in Advances in Service-Oriented and Cloud Computing, Communications in Computer and Information Science, 393 (2013), pp. 1–12.

[9] G. Di Modica, O. Tomarchio, and L. Vita, *A P2P based architecture for Semantic Web Service discovery*, International Journal of Software Engineering and Knowledge Engineering, 21 (2011), pp. 1013–1035.

[10] L. Gurgen, C. Roncancio, C. Labbé, A. Bottaro, and V. Olive, *Sstreamware: a service oriented middleware for heterogeneous sensor data management*, in Proceedings of the 5th international conference on Pervasive services, ACM, 2008.

[11] S. Hadim and N. Mohamed, *Middleware: Middleware challenges and approaches for wireless sensor networks*, IEEE Distributed Systems Online, 7 (2006), pp. 1–.

[12] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, *Middleware to support sensor network applications*, Network, IEEE, (2004).

[13] IEEE Network Working Group, *JavaScript Object Notation (JSON)*, 2006.

[14] V. Issarny, N. Georgantas, S. Hachem, A. Zarras, P. Vassiliadist, M. Autili, M. A. Gerosa, and A. B. Hamida, *Service-oriented middleware for the Future Internet: state of the art and research directions*, Journal of Internet Services and Applications, 2 (2011), pp. 23–45.

[15] K. Khedo and R. K. Subramanian, *Meeca: Misense energy efficient clustering algorithm*, in Wireless Communication and Sensor Networks, 2007. WCSN '07. Third International Conference on, 2007.

[16] X. Koutsoukos, M. Kushwaha, I. Amundson, S. Neema, and J. Sztipanovits, *Oasis: A service-oriented architecture for ambient-aware sensor networks*, in Composition of Embedded Systems. Scientific and Industrial Issues, F. Kordon and O. Sokolsky, eds., Springer Berlin Heidelberg, 2007.

[17] P. Levis and D. Culler, *Mate: a tiny virtual machine for sensor networks*, in Proceedings of the 10th international conference on Architectural support for programming languages and operating systems, ACM, 2002, pp. 85–95.

[18] S. Li, S. Son, and J. Stankovic, *Event detection services using data service middleware in distributed sensor networks*, in Information Processing in Sensor Networks, F. Zhao and L. Guibas, eds., Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2003.

[19] T. Liu and M. Martonosi, *Impala: a middleware system for managing autonomic, parallel sensor systems*, SIGPLAN Not., (2003).

[20] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac, *Internet of things: Vision, applications and research challenges*, Ad Hoc Networks, 10 (2012), pp. 1497 – 1516.

[21] N. Mohamed and J. Al-Jaroodi, *A survey on service-oriented middleware for wireless sensor networks*, Service Oriented Computing and Applications, 5 (2011), pp. 71–85.
[22] L. Mottola and G. P. Picco, *Programming wireless sensor networks: Fundamental concepts and state of the art*, ACM Comput. Surv., 43 (2011), pp. 19:1–19:51.
[23] OGC, *Sensor Web Enablement (SWE)*, 2013. Available at http://www.opengeospatial.org/ogc/markets-technologies/swe/.
[24] M. P. Papazoglou and W.-J. van den Heuvel, *Service Oriented Architectures: approaches, technologies and research issues*, VLDB Journal, 16 (2007), pp. 389–415.
[25] C. Srisathapornphat, C. Jaikaeo, and C.-C. Shen, *Sensor information networking architecture*, in Parallel Processing, 2000. Proceedings. 2000 International Workshops on, 2000, pp. 23–30.
[26] R. Sugihara and R. K. Gupta, *Programming models for sensor networks: A survey*, ACM Trans. Sen. Netw., 4 (2008), pp. 8:1–8:29.
[27] MathML, *MathML(W3C)*, 2013. Available at http://www.w3.org/Math/.

Edited by: Maria Fazio and Nik Bessis
Received: Nov 2, 2013
Accepted: Jan 10, 2014

DOI 10.12694/scpe.v14i4.928
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. 201–213. http://www.scpe.org

A MESSAGE ORIENTED MIDDLEWARE FOR CLOUD COMPUTING TO IMPROVE EFFICIENCY IN RISK MANAGEMENT SYSTEMS*

MARIA FAZIO, ANTONIO CELESTI, ANTONIO PULIAFITO, AND MASSIMO VILLARI †

Abstract. Transportation of Dangerous Goods represents a sensitive problem due its congenital high potential risk of causing disaster if an accident occurs. Transportation of Dangerous Goods Risk Management systems reduce the possibility of both accidental disasters and terrorist attacks detecting unusual events and blocking possible threats. Cloud computing can facilitate the development of such kinds of systems thanks to new emerging paradigms and technologies. In this paper, we discuss the design of a new Message-Oriented Cloud Middleware for Cloud, that can be used to develop a Cloud-based Transportation of Dangerous Goods Risk Management system. More specifically, we investigate issues on Transportation of Dangerous Goods, in order to focus the attention on the requirements of the Risk Management system. Then, we describe how to use the Message-Oriented Cloud Middleware for Cloud architecture and the necessary utilities in particular here for supporting Transportation of Dangerous Goods.

Key words: message oriented middleware, cloud computing, federation, service provisioning, planetary system model.

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction. Risk Management Systems are very complex distributed systems in which different heterogeneous infrastructures and resources need to be properly integrated and managed. In particular, the risks involved in the Transportation of Dangerous Goods (TDGs) over multi-modal ways (e.g., freeways, railways, air and sea routes) have been attracting great interest in the recent years. In fact, dangerous goods can cause terrible disaster if an accident occurs, producing uncontrollable effects in highly populated areas or during popular events. Moreover, the risk becomes more concrete if we consider that dangerous goods can potentially be an objective of terrorist attacks. TDGs is a very complex problem, involving economical, legislative and technological aspects. The complexity raises due to the fact that for reducing as much as possible all risks, the previous aspects need to be addressed all together.

Nowadays, advanced technologies in the field of ICT (Information and communications technology) promise a way to track in real time the entity of such transportations and efficiently manage the exposure to related risks. Innovative technologies can actively support goods tracking and provide valuable added value services to provide legally requested information and also to minimize risk in case of failures and accidents. Nevertheless, the development of a TDG Risk Management System is not easy at all due to the number of utilities that need to be integrated and coordinated (e.g., sensing, high performance computing, storage, security, etc).

Cloud computing has reached a high level of complexity embracing many application fields. Indeed, the Cloud-like technologies allow the development of next generation versatile systems in which different types of technologies and hardware/software solutions can be integrated.

In this paper, we present a novel Message-Oriented Middleware (MOM4C), that can be usefully adopted for the development of a TDG Risk Management System. MOM4C allows to set up Cloud facilities aggregating different Cloud utilities coming from different enterprises, organizations, and governments in a federated environment. According to the MOM4C terminology, a "Cloud Facility" is a mash-up Cloud service composed integrating one or more Cloud utilities, instead, a "Cloud utility" is a specific Cloud service (e.g., virtualization, storage, network, computation, security, sensing, data analytics, etc). In simple words, the aim of the middleware is to acts as a liaison among utilities in order to support the deployment of advanced, flexible, and differentiated Cloud facilities [17]. In such a versatile scenario enterprises, organizations, and governments become, at the same time, customers and providers. MOM4C provides flexibility, efficiency, and elasticity for the setup of Cloud facility to Cloud providers, seamlessly integrating the utilities belonging to different heterogeneous environments or administrative domains. It allows to expand existing Cloud systems and to integrate several virtual and physical resources. Its ability of collecting heterogeneous utilities and abstracting their functionalities to high level Cloud facilitates is very useful for the development of advanced applications

*This work was partially supported by Projects SIMONE and SIGMA, Italian National Operative Program (PON) 2007-2013.
†DICIEAMA, University of Messina, Contrada Di Dio, 98166 Sant'Agata - Messina (mfazio(acelesti,apuliafito,mvillari).unime.it).

for Internet of Things (IoTs). MOM4C has been designed according to the Message-Oriented model. This model has already been used for the designing of Cloud middleware such as IBM WebSphere MQ (MQSeries), TIBCO Rendezvous, and RabbitMQ. In comparison with them, MOM4C allows to develop services fitting the requirements of Cloud computing.

Due to its features, MOM4C can offer a solid support to TDG scenarios. A TDG Risk Management System mainly requires: i) a monitoring system able to localize and track dangerous goods even analyzing their states according to different types of information (e.g, temperature, pressure, gas detection, etc); ii) a data collection and elaboration system able to correlate the different pieces of information coming from the monitoring system; iii) an intelligence transportation system able to provide: transport mode optimization and traffic management through a "smarter" use of transport networks; iv) an informative system able to disseminate alerts to the population in case of disaster providing pieces of information that potentially can save lives.

In order to satisfy such requirements, we analyze the possibility of arranging Cloud facilities for TDG Risk Management Systems (TDG Cloud Facilities) combining several Cloud utilities, in particular we gathered the utilities we develop next, in four main branches: sensing, virtualization, big data management, and trusted computing.

The rest of the paper is organized as follows. In Sect. 2, we discuss the main concerns regarding the TDG. In Sect., 3, we present the MOM4C computing model, discussing a few architectural aspects in Sect. 4. An example of TDG Cloud facility arranged be means of MOM4C is discussed in Sect. 5. A possible combination of both hardware/software solutions and technologies for the implementation of TDG Cloud facilities is discussed in Sect. 6. In Sect. 7, we provide an overview regarding other available Cloud middleware, highlighting how they differ from MOM4C. Conclusions and remarks are summarized in Sect. 8.

2. TDG Concerns. TDG risk management systems able to reduce the risk of both accidental disasters and terrorist attacks make extensive use of sensing infrastructures to assess the risk itself and to detect unusual events. TDG risk management systems asks for a continuous monitoring of activities related to transportation. It is necessary not only to track the position of the vehicle and the status of the cargo, but it is also important to understand how the environment interacts during the transportation of dangerous goods. Automatic vehicle identification techniques relying on Radio Frequency Identification (RFID) permit to electronically gather shipment information. Route planning can reduce the probability of disaster. It can be time-independent or reactive. In particular, route planning is reactive if real-time pieces of information about the conditions of the transport network are periodically updated in the management system. Such pieces of information are gathered by sensor networks and made available in real-time databases. In addition, Geographic information System (GIS) will permit geospatial data management for decision making processes.

2.1. The State of the Art on TDG. The TDG problem has been gathering great attention from both research community and business companies. The main goal is to develop a TDG risk management systems able to prevent disasters. In ICT fields, several initiatives appeared, each one addressing specific requirements.

MITRA [2] is a research project funded by the European Commission with the objective to prototype a new operational system based on regional responsibilities for the monitoring of dangerous goods transportation in Europe. It provides a real-time knowledge of position and contents of dangerous goods through the European Geostationary Navigation Overlay Service (EGNOS), that is a satellite based augmentation system developed by the European Space Agency, the European Commission and EUROCONTROL. In case of dangerous situations, GSM communications allow to alert the Security Control Centre, which is responsible to prevent accidents, manage crisis and enable quick intervention.

SMARTFREIGHT [3] is a European research project, partly funded by the European Commission under the 7th Framework Program (7FP). The overall objective of SMARTFREIGHT is to address new traffic management measures towards individual freight vehicles by using open ICT services, with an emphasis on the interoperability between traffic management and freight distribution systems, and an integrated heterogeneous wireless communication infrastructure within the framework of CALM (Communication Access for Land Mobiles)

In [18], the authors propose a complete monitoring and tracking solution for truck fleets. The system exploits battery-powered environmental sensors (temperature, humidity, pressure, gas concentration and ionizing radiation levels), connected by a ZigBee-based Wireless Sensor Network. Collected data is then sent from the

vehicle to a remote server via a GPRS link. The GPS positioning system is integrated by the use of an Inertial Navigation System, which guarantees a precise estimate of the position also when the GPS signal is weak or temporarily lost.

The solution proposed in [20] aims to improve the security of maritime container transport of dangerous goods by the real-time monitoring of container state. This system uses micro-sensor technologies and radio frequency communication technology to obtain the dangerous goods condition inside containers, as well as automatic positioning in the cargo hold. Information on the state of dangerous goods are transmitted to the shore monitoring center on land through INMARSAT stations.

By comparing the different solutions for dangerous goods transportation, we have identified the following common goals: 1) localization and tracking means of freight transportation, 2) monitoring of goods according to several types of information (temperature, pressure, gas detection,...), 3) data collection and elaboration, 4) definition of policies for disaster prevention, 5) definition of policies for emergency management. However, the existing solutions exploit heterogeneous systems, hardly to be integrated. Indeed, they differ a lot in terms of sensor technologies, communication infrastructures, design of the system organization and software support. Here, our idea is to setup an environment able to harmonize these heterogeneous systems.

2.2. Open Issues. Companies operating in the monitoring of dangerous goods have to use specific technologies that depend on several factors: the type of dangerous goods that are tracked, their geographical position and route, means of transport, legislation of the country and so on. International Regulations define standard procedures for the treatment of dangerous goods. However, from a technological point of view, they do not provide any specification with reference to the monitoring infrastructure installation. The result is that actually there is no compatibility between different monitoring systems managed by organizations or companies, both in terms of hardware and software.

Another important issue is related to the transportation of the adopted solution. Each solution focuses on a specific method of transportation (such as ship, truck, airplane or railways) and the concept of multi-modal service is not faced at all. However, the aggregation of information from multi-modal ways can be extremely useful to predict terrorist attacks. Furthermore, in case of attacks, the management of different types of way out from the disaster area can save human lives.

A world wide standardized solution is still missing. Recent events have shown the importance of collaboration among different countries to fight against terrorism. So, we imagine a future transportation system where efforts will integrate activities along the roads, highways, railways, harbors and airports at once. The integration will also include activities provided by different operators inside the same country and among different countries.

3. The MOM4C Computing Model. Currently, many pieces of Cloud middleware have been appearing on the market. As highlight on the state of the art analysis discussed in Sect. 7, the available solutions are related to specific scenarios. On the contrary, the TDG risk management system requires to address a versatile scenario in which different utilities have to be integrated (e.g., sensing, virtualization, big data management, trusted computing, etc). For such a reason, in this paper we present a solution based on the MOM4C, a solution that in our opinion, well fits the requirements of TDG risk management systems. Differently from other available pieces of middleware, analyzed in Sect. 7, MOM4C abstracts the type of offered services, providing a framework able to integrate both the current and future Cloud solutions, offering to the customers the possibility to customize their Cloud facilities.

3.1. The Need of a Middleware for Emerging Cloud-Based Systems. Analyzing the trend of the Cloud computing market, we can highlight, on one hand, a growing number of providers that are investing in Cloud-based services and infrastructures and, on the other hand, the interest of companies in long-term, customizable and complex business solutions, which must be easy to be set up, reliable, and accessible through the Internet. MOM4C has been design to fill up this gap, integrating existing infrastructures and resources in form of Cloud utilities into one efficient, scalable, reactive and secure distributed system. Its deployment can be strategic for many different stakeholders, as shown in Fig. 3.1. MOM4C enables third-party enterprises and developers to implement Cloud facilities in an easy way, integrating different Cloud utilities (e.g., storage, network, computation, security, sensing, data analytics, etc) according to a mash-up development model. In

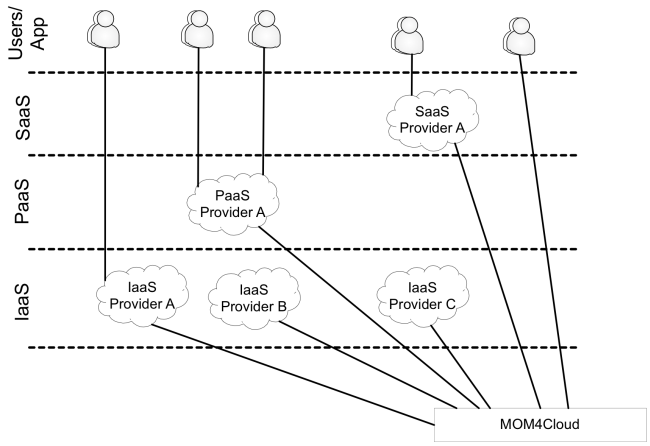


FIG. 3.1. *All stakeholders and cloud layers involved in MOM4C reference scenarios.*

this way, enterprises, organizations, and governments can quickly Cloud facilities integrating different Cloud utilities.

MOM4C enables Cloud providers to abstract the service level. Typically, Cloud providers can deliver three main service levels, i.e., Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). According to such a classification, MOM4C allows to develop Cloud facilities in form of IaaS, PaaS, and SaaS instances. It is important to notice that also Cloud utilities themselves can be hardware/software functionality delivered in form of IaaS, PaaS, and SaaS.

IaaS Providers deliver computers and devices (i.e., physical and/or virtual) and other resources. Typically, a Virtual Infrastructure Manager (VIM) controls one or more hypervisors each one running several Virtual Machines (VMs) as guests. A VIM allows to manage a large numbers of VMs (e.g., preparing disk images, setting up networking, starting, suspending, stopping VM, etc) and to scale services up/down according to customers' requirements. An example is represented by a provider that offers to end-users on-demand VMs execution. PaaS providers deliver a computing platform, typically including operating system, programming language execution environment, database, and web server. Software developers can implement and run their solutions on a Cloud platform without the cost of buying and managing the underlying hardware and software layers. Typically, the underlying computer and storage resources automatically scale up/down to match application demand. Another example is represented by a provider that offers a platform that collects data coming from one or more sensor networks and that offer Application Program Interfaces (APIs) for data processing, hence enabling developers to implement intelligent sensing applications. SaaS providers, typically deliver on-demand pieces of software via Web 2.0 that are usually priced on a pay-per-use basis. Providers install and manage applications in the Cloud and users access these ones from software clients, generally web browsers. A case in which a provider that offers via Web 2.0 interface an office automation software suite such as Google Drive to manage documents. Furthermore, a Cloud facilities built through MOM4C will be able to integrate Cloud utilities even belonging to different administrative domains in a federated system. In a federation, each entity is independent and can not be conditioned by a "central government" in its activities. The components of a federation are in some sense "sovereign" with a certain degree of autonomy from the "central government": this is why a federation can be intended more than a mere loose alliance of independent entities. Moreover, the treatment of all the data and information transferred through MOM4C is performed according to secure policies able to assure: data confidentiality, data integrity, data authenticity, non-repudiation of the sender, non-repudiation of the receiver.

3.2. MOM4C: a Planetary System Model. The MOM4C computing model was inspired by a planetary system model. Due to its native ability in integrating heterogeneous infrastructures and resources in form of Cloud utilities, MOM4C can potentially offer a wide plethora of Cloud Facilities able to provide complex, customizable and differentiated mash-up services.

A monolithic design of the proposed system is inconceivable, since it implies a heavy effort in management

of all the available components, low scalability and useless service availability for clients. On the contrary, to guarantee the maximum flexibility, we have conceived MOM4C as a very modular architecture, in which every client can customize Cloud facilities according to their business requirements. From the client point of view, we can schematize MOM4C as well as a planetary system, as shown in Fig. 3.2. The planetary system is composed

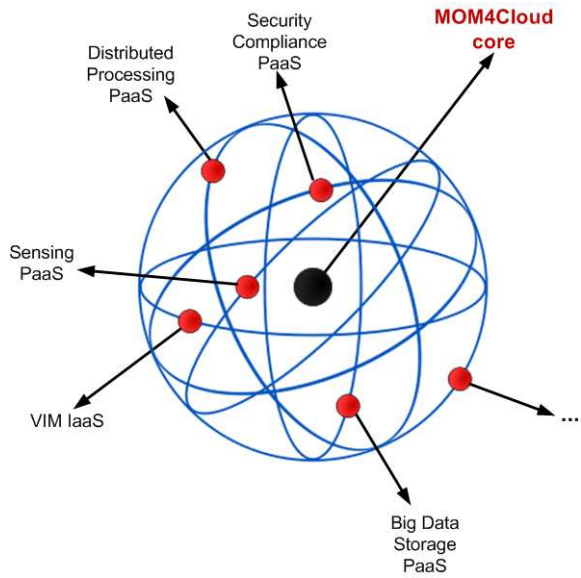


FIG. 3.2. *Planetary system model for service provisioning through MOM4C.*

by one or more planets that orbit around a central star. According to our abstraction, Planets identify available utilities. For example, utilities can be: i) VIM, for on-demand VM provisioning; ii) Sensing PaaS, collecting data by different sensing environments; iii) Distributed Processing PaaS, providing high computational power; iv) Big Data Storage PaaS, providing distribute storage for huge amount of data, and so on. The core of MOM4C is the star of the planetary system. It provides all the basic functionalities necessary for the life of planets. Specifically, it includes a scalable messaging and presence system, security mechanisms for data integrity, confidentiality and non-repudiation, federation management and other specific communication features for the management and integration of heterogeneous utilities.

All the possible combinations of planets specialize the behavior of the planetary system. According to our similitude, a specific planetary system configuration, including target planets defines the Cloud facility. In fact, according to our definition, the Cloud facility has to be customizable from clients in order to fit specific business scenarios.

4. MOM4C Architecture. MOM4C is designed according to the message-oriented paradigm, in order to provide an efficient communication system among different distributed components. From the message-oriented paradigm, MOM4C inherits a primary benefit, that is loosing coupling between participants in a system due to their asynchronous interaction. It results in a highly cohesive, decoupled system deployment. It also decouples the performance of the subsystems from each other. Subsystems can be independently scaled, with little or no disruption of performance into the other subsystems. With reference to the management of unpredictable activity overloads in a subsystem, the message-oriented model allows to accept a message when it is ready, rather than being forced to accept it. MOM4C adds important features, that are strategic for business in Cloud. Its major benefits includes:

- **Modularity**: the middleware can be quickly extended using different modules characterizing different available utilities. It can be easily customized in order to suit a specific Cloud scenario.
- **Polymorphism**: each distributed entity in the system can play different roles according to the system requirements. Different rules includes both the core management tasks and the utility-related tasks.

- **Security**: an indispensable requirement for the large-scale adoption Cloud computing is security, especially in business scenarios. Security has to be natively addressed at any level of communication (intra-module, inter-module, and inter-domain), providing guarantees in terms of data confidentiality and data integrity.
- **Federation**: it is a strategic approach to promote collaboration among cooperating Cloud providers.

4.1. Cluster and Execution Layers. As depicted in Fig. 4.1, MOM4C is based on a distributed architecture, organized in two layers, that are the Cluster Layer (CL) and the Execution Layer (EL). The Cluster

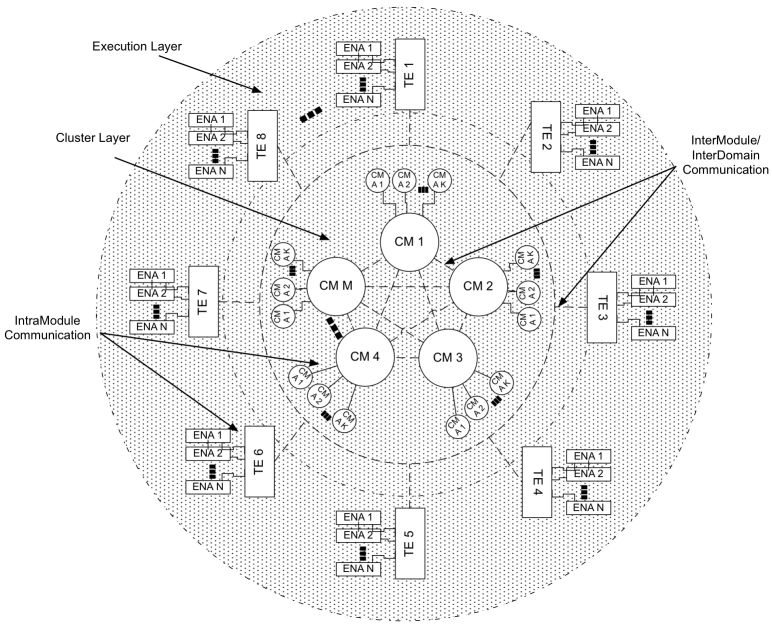


FIG. 4.1. *MOM4C basic scheme.*

Layer represent the "core" of MOM4C. It consists of an overlay network of decentralized Cluster Manager (CM) nodes. Each CM is responsible for the working activities of the Task Executor (TE) nodes belonging to the cluster. The EL is composed of TEs, which are intended to perform operative tasks. TEs can be trained to perform a specific task. It means that they do not instantiate all the services and utilities available in MOM4C, but they download code, initialize and configure services, launch software agents whenever they receives instructions from the CM. An appropriate utility module configuration into TEs allows to specialize MOM4C services. According to the specific code in execution at TEs, we have different characterizations of the EL.

To perform different types of tasks (e.g., VM execution and sensing data gathering), we set up specialized ELs, which independently works according to the CL specifications. Such an organization of roles and activities carries out high modularity to the MOM4C system. Building around the Cluster Layer many TE layers at the same time characterizes the MOM4C behavior. Thus, an ad-hoc layers configuration is designed to support a specific scenario. With reference to the planetary system model, the star includes all the functionalities of the Cluster Layer, which sustains the whole system. Any orbit represents a specific Execution Layer and the planet is the utility offered by TEs belonging to the related Execution Layer.

Another important feature of MOM4C is the polymorphic nature of nodes. At different times, each physical node can serve as CM or TE. However, only a node in a cluster is elected as CM and actively works for managing the whole cluster. Some other node are elected as "passive CMs", which are redundant CMs that can quickly replace the active CM if it fails. This approach improves the fault tolerance of the CL. The size of the cluster depends on the system workload and it can dynamically change according to the specific elasticity requirements of the system. About TEs, they can belong to one or more ELs, hence they work at different Cloud utilities. Such a concept is better explained in Fig. 4.2. For example, TE 1, 2, 3, 4, 5, 6 are hypervisor servers working

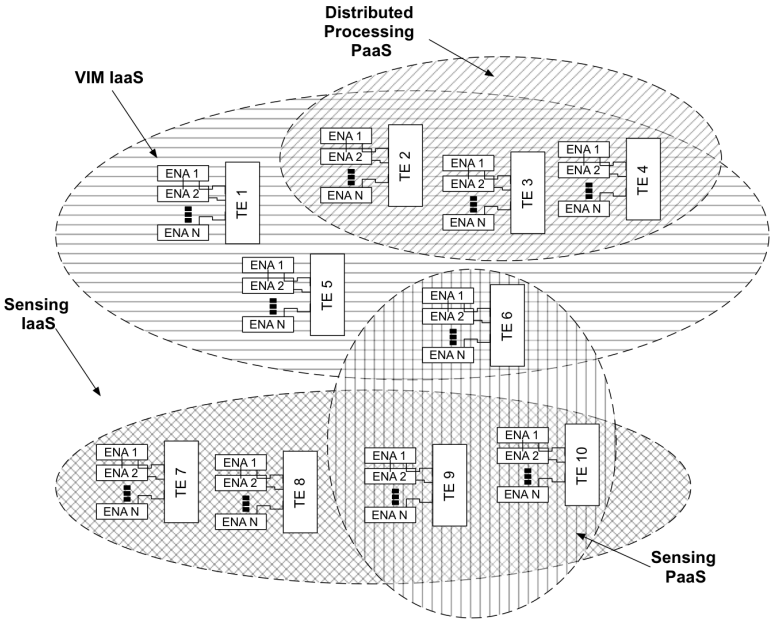


FIG. 4.2. *Hybrid Executor Node Layer composition.*

to provide a VIM IaaS. At the same time, TE 2, TE 3, and TE 4 work also to provide a Distributed Processing PaaS, since software agents running on TEs are independent active processes. Following the example in Fig. 4.2, TE 7, 8, 9, and 10 work as embedded devices for Sensing IaaS provisioning, whereas TE 6, 9, and 10 works for a Sensing PaaS, for example collecting sensing data from TE 9 and 10 and providing services through the AJAX Web APIs of Web application deployed in TE 6.

4.2. All Turn Around the Communication System. The strong point of MOM4C is represented by its communication system. In fact, the middleware supports three types of communications:

- **IntraModule Communication**: it characterizes information exchange inside each node of the architecture, both CMs and TEs. It guarantees a seamless way for allowing their internal software modules to communicate each other.
- **InterModule Communication**: it governs communications between CMs and TEs and vice-versa.
- **InterDomain Communication**: is specific for communications among CMs belonging to different administrative domains, hence enabling InterCloud or Cloud federation scenarios.

In order to ensure as much as possible the middleware modularity, the tasks running on each node are mapped on different processes within the Operating System, which communicate each other by means of an Inter Process Communication (IPC) or InterModule communication. According to the message-oriented design of MOM4C, InterModule communications are based on an Instant Messaging and Presence (IMP) protocol. A presence system allows participants to subscribe to each other and to be notified about changes in their state. On the other hand, Instant messaging is defined as the exchange of content between a set of participants in near real time. InterDomain communications among different administrative domains are managed considering the federation agreements among the domains. Federation allows Cloud providers to "lend" and "borrow" resources. Thus, a CM of a domain is able to control one or more TEs belonging to other domains.

5. A Cloud Facility for TDG Risk Management Systems. In this Sect., we firstly discuss what the requirements are for a TDG Risk Management System, and than, we present and example of Cloud facility for TDG Risk Management System (TDG Cloud facility) combining four Cloud utilities, i.e., sensing, virtualization, big data management, and trusted computing.

5.1. Functional and Non-Functional Requirements. By comparing the different available initiatives in the field of TDG, analyzed in Sect. 2.1, we have identified the following functional requirements:

1. monitoring, localizing and tracking of dangerous goods even analyzing their states according to different types of information (e.g, temperature, pressure, gas detection, etc);
2. collect, analyze, and correlate the different pieces of information coming from different monitoring activities
3. transport mode optimization and traffic management through a "smarter" use of transport networks
4. disseminate alerts to the population in case of disaster providing pieces of informations that potentially can save lives.

Cloud computing can offers several benefits in carrying out all these activities and we are going to explain how by means of MOM4C.

Considering the monitoring related to the transportation of dangerous goods, existing solutions differ a lot in terms of sensor technologies, communication infrastructures, design of the system organization and software support. Companies operating in the monitoring of dangerous goods have to use specific technologies that depend on several factors: the type of dangerous goods that are tracked, their geographical position and route, mode of transport, legislation of the country and so on. International Regulations define standard procedures for the treatment of dangerous goods. However, from a technological point of view, they do not provide any specification with reference to the monitoring infrastructure installation. The result is that, usually, there is not compatibility between different monitoring systems managed by different organizations or companies, both in terms of hardware and software. Another important point is related to the adopted transportation solution. Each solution focuses on a specific method of transportation (such as ship, truck, airplane, or railways) and the concept of *concurrent* service is not faced at all. However, the aggregation of pieces of information from *concurrent* ways can be extremely useful to predict terrorist attacks. Furthermore, in case of attack, the management of different types of way out from the disaster area can save human lives.

Regarding non-Functional requirements, the TDG Cloud facility has to abstract the underlying infrastructures and resources through Cloud utilities. In fact, the MOM4C, represents the *GLUE* to integrate and homogenize such heterogeneous infrastructures and resources. By using the concept of virtualization, the MOM4C can abstract hardware and software resources and, thus, guarantee an high level of interoperability among different physical infrastructures involved in the intelligent transportation activities.

The monitoring activity causes a massive collection of data, which need to be organized and processed in a transparent way, in order to provide an integrated knowledge of the context. The context knowledge is the base to build up strategies at the National Security level. High amount of data means more efficient services, but implies high requirements in terms of processing power and storage space. However, the demands of resources significantly vary depending on several parameters, such as the geographical area, traffic, and so on. The distributed nature of Cloud computing guarantees high availability of computational and storage resources as services, which can be dynamically adapted to specific needs of the system. Concurrent transport of dangerous goods is characterized by specific constrains, which need guarantees on the quality of the informative services (e.g., reaction time to an event occurrence, synchronization of activities, trust in using third party support, etc). The high flexibility of the Cloud in dynamic configuration, management and optimization of resources and services allows to effectively respond to the quality of service requirements of the system.

Smart services supporting the transport requires to correlate pieces of information regarding the environment, goods, carriers and freight operators, and determine the best routes for goods transfer. MOM4C offers a very innovative approach to develop TDG risk management system through a distributed system where resources and context information are accessible through well-defined interfaces. This approach allows to implement new services without any knowledge of physical infrastructures and software architecture, making the TDG easier and flexible. Another important feature offered by MOM4C is its ability to manage a federation of several cloud providers (i.e., enterprises, organizations, and governments). Thus, we consider the Cloud environment as a constellation of hundreds of independent, heterogeneous, private/hybrid Clouds able to interact each other, maintaining separated their own administration domains accomplishing *inter-cloud* scenarios. This requirement is particularly important in the transport management, because actors that interact to improve their services do not intend to disclose their informative systems. Another important requirement in case of terroristic attack, is

the security of the TDG risk management system. In fact, in order to avoid "man in the middle" attacks, causing potential data corruption or unauthorized information disclosure, both communications among the different components and the access to these latter have to be trusted.

5.2. MOM4C Utility Composition. Thanks to its modularity, MOM4C allows to instantiate different types Cloud facilities. As previously discussed, as well as a planetary system is composed by a star with several planets that turn around it along their orbits, in MOM4C, a Cloud facility is built around the MOM4C core (i.e., the central star) and several Cloud utilities (i.e., planets). From an architectural point of view, we remark that the MOM4C core consist of an overlay network of decentralized CM nodes, whereas a each Cloud utility consists of an overlay network of TE nodes that offer a particular service.

In order to better explain the planetary system model at the basis of the MOM4C design, let us consider the possible utility composition to support the TDG risk management scenario. Considering both the functional and non-functional requirements discussed in Sect. 5.1, in our opinion a possible TDG Cloud facilities arranged with the MOM4C should four main Cloud utilities: sensing, virtualization, big data management, and trusted computing.

The *sensing* utility allows to virtualize different types of sensing infrastructures, adding new capabilities in data abstraction. It gathers sensing information from a peripheral decision-maker, called Virtual Pervasive Element (VPE), able to interact with smart sensing devices or sensing environments [10]. The *VIM* utility allows to aggregate heterogeneous computing infrastructures, providing suitable interfaces at the high-level management layer for enabling the integration of high-level features, such as public Cloud interfaces, contextualization, security [6] [12] and dynamic resources provisioning [8]. The *big data management* utility allows to storage a huge amount of data an to perform an efficient retrieval of them adopting, for example, the map/reduce approach. The *Trusted Computing utility* allows interact with the Trusted Platform Module (TPM) on the physical host [7] by means of a software agent. The TPM is a hardware micro-controller that allows to combine hardware and software components by building a chains of trust. In addition by means of the remote and deep attestation protocols, the utility is able to verify the configuration of physical hosts and VMs.

Figure 5.1 depicts an example of TDG Cloud facility combining seven Cloud utilities. The utilities are orchestrated by the MOM4C core with which communicate in a secure way through the MOM4C communication system. Utility 1 collects sensing data coming from several sensor networks monitoring different transport ways (i.e., freeways, railways, air and sea routes). Utility 2, add to the Cloud facility the ability to virtualize the physical datacenter, by means of a VIM, in order to arrange different scalable virtual environments. In addition, this utility allows the Cloud facility to scale up/down the virtual infrastructure asking external resources when it is required (e.g., when the physical resources are run out the Cloud facility can ask for resources to external providers). Utility 3 enables big data management through a system able to store huge amount of pieces of data and to efficiently retrieve and process them using map-reduce mechanisms. The utility is built into a virtual infrastructure that can be scaled up/down when required thanks to utility 2. Finally, utility 4 adds to the Cloud facility trusted computing capabilities enforcing remote attestation in both physical and virtual servers respectively considering physical and virtual TPMs. In this way, if a physical or virtual machine is corrupted the utility will be able to immediately detect the attempt of attack and block it.

Regarding the secure communication between the MOM4C core and the various utilities, the middleware natively involve secure communication by means of digital signing and message encryption mechanisms. From an architectural point of view, this means that both CM and TE nodes communicate each other through secure channels. Such a feature is enforce by MOM4C independently from any specific type of Cloud facility.

6. Solutions and Technologies for the TDG Cloud Facility.

6.1. Communication System. According to the design specifications of MOM4C, the InterModule and InterDomain communication systems have been implemented using a well known MIPO, that is XMPP. The XMPP (RFC 3920 and RFC 3921), also called Jabber, is becoming more and more popular due to its flexibility to suit different scenarios where a high level of re-activeness is strongly required. Despite it was born for human interaction via chat room it can be used to develop the communication of whatever distributed system well fitting the requirements of Cloud computing. XMPP is an XML-based protocol used for near-real-time, extensible instant messaging and presence information. XMPP remains the core protocol of the Jabber Instant

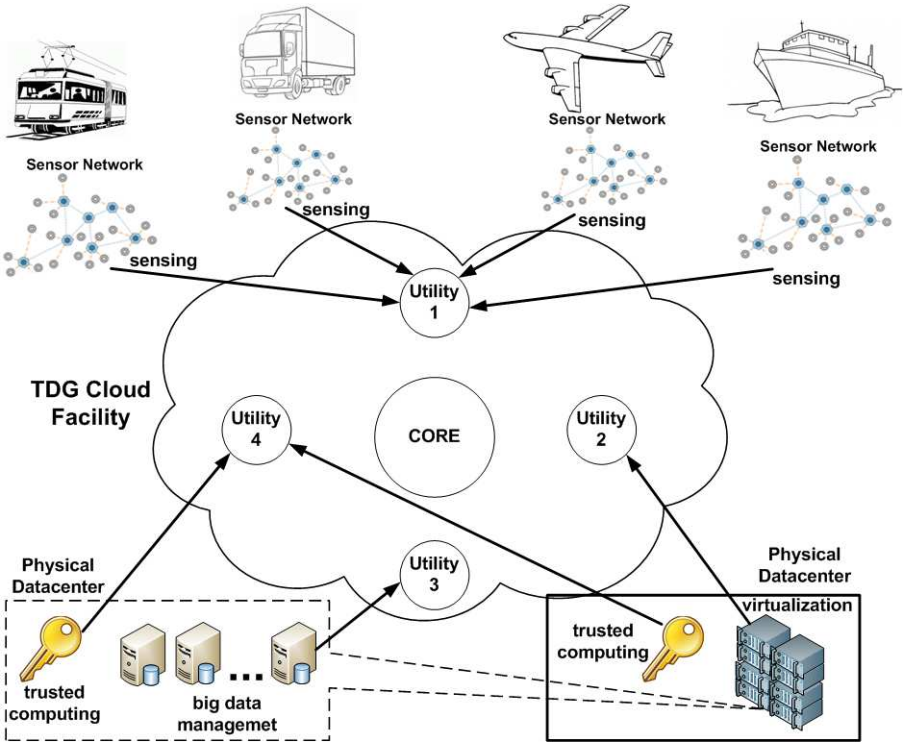


FIG. 5.1. *Example of TDG Cloud facility with MOM4C.*

Messaging and Presence technology. The "Jabber" technology leverages open standards to provide a highly scalable architecture that supports the aggregation of presence information across different devices, users and applications. As the client uses HTTP, most firewalls allow users to fetch and post messages without hindrance. Thus, if the TCP port used by XMPP is blocked, a server can listen on the normal HTTP port and the traffic should pass without problems. Custom functionality can be built on top of XMPP, and common extensions are managed by the XMPP Software Foundation. This make the protocol flexible to be extended with QoS and security features.

6.2. Cluster Manager Node. The CM Coordinator is the core of a CM.In order to communicate with other nodes of the system, the CM Coordinator exploits three different interfaces: IntraModule Interface, used to interact with specific modules, InterDomain Interface, used for interconnecting the CMs belonging to different administrative domain, TE Interface, for communicating with all the TE nodes. Apart from the IntraModule Interface that works by means of the DBUS communication system, the InterModule and InterDomain interfaces are connected to different XMPP rooms, in order to separate different communications. It performs high level operations assigning tasks to different TEs, taking into account the system workload and features of each node. Moreover, through InterDomain Interface, the CM Coordinator provides information about the Cluster state, collected through its Interfaces.

6.3. Task Executor Node. The TE Coordinator is the main component of the TE node. It is responsible to execute the command sent by a CM. In addition, it monitors resources and the Operating System in order to optimize their usage. For example, a real time information on CPU, RAM, storage and network utilization can be acquired. The main activities of a TE Coordinator are: 1) Streaming the collected information; 2) providing the collected information on demand; 3) sending a specific notification (alert) when a pre-determined condition is verified. All the TE Coordinators have to interact exploiting the persistent XMPP connection made available through the CM Coordinator Interfaces. The other nodes, in order to perform temporary peer-to-peer communications, can attend an ephemeral XMPP session connecting themselves to an "utility room".

6.4. Possible Solutions for Utility Implementation. In Sect. 1, for arranging Cloud facilities useful for TDG Risk Management Systems, we recognized four types of utility: sensing, virtualization, big data management, and trusted computing. The implementation of these Cloud utilities can be achieved using different software tools and frameworks. Here below we describe what we selected for accomplishing our cloud environment.

The sensing utility, has to be developed considering the Sensor Web Enablement (SWE) standard which enables developers to make all types of sensors, transducers and sensor data repositories discoverable, accessible and useable via the Web. Further standards that have to be considered includes

- **Observations & Measurements (O&M)**. Standard models and XML Schema for encoding observations and measurements from a sensor, both archived and real-time.
- **Sensor Model Language (SensorML)**. Standard models and XML Schema for describing sensors systems and processes associated with sensor observations; provides information needed for discovery of sensors, location of sensor observations, processing of low-level sensor observations, and listing of task-able properties, as well as supports on-demand processing of sensor observations.
- **Transducer Model Language (TransducerML or TML)**. The conceptual model and XML Schema for describing transducers and supporting real-time streaming of data to and from sensor systems.
- **Sensor Observations Service (SOS)**. Standard web service interface for requesting, filtering, and retrieving observations and sensor system information. This is the intermediary between a client and an observation repository or near real-time sensor channel.
- **Sensor Planning Service (SPS)**. Standard web service interface for requesting user-driven acquisitions and observations. This is the intermediary between a client and a sensor collection management environment.
- **Sensor Alert Service (SAS)**. Standard web service interface for publishing and subscribing to alerts from sensors.
- **Web Notification Services (WNS)**. Standard web service interface for asynchronous delivery of messages or alerts from SAS and SPS web services and other elements of service workflows.

There are several Open Source, Free-Ware, and Commercial Off-the-Shelf (COTS) activities committed to the development of Sensor Web Enablement (SWE) oriented software. This includes software to support servers, middleware, and clients, as well as tools for creating and validating SWE encodings. Interesting tools include 52 North, MapServer, OOSTethys, Space Time Toolkit, SWE Common Library, Process Execution Engine Library,

The virtualization utility can be developed using different hypevisors and frameworks (e.g., KVM/QEMU, XEM, VMware, VirtualBox, libvirt, etc). Regarding the VIM, developers have to consider the possibility either to develop a customize solution or using existing solution including, for example, OpenStack, Open Nebula, Clever, Nimbus, Eucalyptus, etc. In addition the Open Virtualization Format (OVF) standard has to be considered.

The big data management utility can be developed considering both a distributed file system and a parallel processing system able to fast retrieve and process pieces of data. To this regard, a possible solution is represented by Apache Hadoop. It is a popular framework providing both a distributed file system (HDFS), and a processing environment adopting the map-reduce paradigm. Further valuable alternative solutions include Nutch, Cloudera, Hypertable, HBase, Apache Mahout, and Apache Cassandra.

Regarding the trusted computing utility, the Institute for Applied Information Processing and Communication (IAIK) of the Graz University of Technology (AT) have been developing many software libraries and tools. The Trusted Computing Group (TCG) has defined a Trusted Software Stack (TSS) to simplify the access from software modules to TPM. In particular TSS defines an Application Programming Interface to operating systems and applications. Furthermore for supporting the development of trusted applications, the TGG has defined TCG Device Driver Library (TDDL). Further details are available in [1]

7. Related Works. Some works in literature deal with the need of Cloud middleware, addressing specific issues and exploiting different technologies. To support application execution in the Cloud, in [13], authors present CloudScale. It is a piece of middleware for building Cloud applications like regular Java programs and easily deploy them into IaaS Clouds. It implements a declarative deployment model, in which application developers specify the scaling requirements and policies of their applications using the Aspect-Oriented Pro-

gramming (AOP) model. A different approach is proposed in [19]. The authors present a low latency fault tolerance middleware to support distributed applications deployment within a Cloud environment. It is based on the leader/follower replication approach for maintaining strong replica consistency of the replica states. If a fault occurs, the reconfiguration/recovery mechanisms implemented in the middleware ensure that a backup replica obtains all the information it needs to reproduce the actions of the application. The middleware presented in [5] has been designed aiming mission assurance for critical Cloud applications across hybrid Clouds. It is centered on policy-based event monitoring and dynamic reactions to guarantee the accomplishment of "end-to-end" and "cross-layer" security, dependability, and timeliness. In [9], the authors present a piece of middleware for enabling media-centered cooperation among home networks. It allows users to join their home equipments through a Cloud, providing a new content distribution model that simplifies the discovery, classification, and access to commercial contents within a home network. In [14], the authors focus their work on the integration of different types of computational environments. In fact, they propose a lightweight component-based middleware intended to simplify the transition from clusters, to Grids and Clouds and/or a mixture of them. The key points of this middleware are a modular infrastructure, that can adapt its behavior to the running environment, and application connectivity requirements. The problem of integrating multi-tenancy into the Cloud is addressed in [4]. The authors propose a Cloud architecture for achieving multi-tenancy at the SOA level by virtualizing the middleware servers running the Service Oriented Architecture (SOA) artifacts and allowing a single instance to be securely shared between tenants or different customers. The key idea of the work is that the combination between virtualization, elasticity and multi-tenancy makes it possible an optimal usage of data center resources (i.e., CPU, memory, and network). A piece of middleware designed for monitoring Cloud resources is proposed in [16]. The presented architecture is based on a scalable data-centric publish/subscribe paradigm to disseminate data in multi-tenant Cloud scenarios. Furthermore, it allows to customize both granularity and frequency of received monitored data according to specific service and tenant requirements. The work proposed in [11] aims to support mobile applications with processing power and storage space, moving resource-intensive activities into the Cloud. It abstracts the API of multiple Cloud vendors, thus providing a unique JSON-based interface that responds according to the REST-based Cloud services. The current framework considers the APIs from Amazon EC2, S3, Google and some open source Cloud projects like Eucalyptus. In [15], the authors present a piece of middleware to support fast system implementation and ICT cost reduction by making use of private Clouds. The system includes application servers that run a Java Runtime Environment (JRE) and additional modules for service management and information integration, designed according to a SOA.

8. Conclusion and Remarks. In this paper, we have discussed the design of a Cloud-based Risk Management System for the Transportation of Dangerous Goods (TDG). A TDG Risk Management System requires the integration of different heterogeneous sensing infrastructures and different ICT assets regarding for example monitoring, processing, storage, etc.

MOM4C enables software architects to seamlessly design such a kind of distributed system thanks to a message oriented approach. In fact, MOM4C, according to a planetary system model, allows software architects to arrange distributed systems as Cloud facilities combining different utilities. In addition, the middleware allows to different enterprises, organizations, and governments to cooperate in a federated Cloud environments in a transparent way.

More specifically, an example of TDG Cloud facility has been described combining four main Cloud utilities, i.e., sensing, virtualization, big data management, and trusted computing. An interesting aspect of MOM4C is its ability to adapt the Cloud facilities to the system requirements even in a heterogeneous environment. As we have demostrated, such a feature makes the middleware, a valuable solution for the development of next generation versatile systems in the field of TDG.

REFERENCES

[1] Trusted Computing Grouop (TCG): http://www.trustedcomputinggroup.org/.
[2] 2004-2006. MITRA: Monitoring and intervention for the transportation of dangerous goods. http://www.mitraproject.info/.
[3] 2009. SMARTFREIGHT project, FP7-216353. http://www.smartfreight.info//.

[4] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, and P. Fremantle, *Multi-tenant SOA Middleware for Cloud Computing*, in IEEE CLOUD'10), 2010, pp. 458–465.

[5] R. Campbell, M. Montanari, and R. Farivar, *A middleware for assured clouds*, Journal of Internet Services and Applications, 3 (2012), pp. 87–94.

[6] A. Celesti, M. Fazio, M. Villari, and A. Puliafito, *Se clever: A secure message oriented middleware for cloud federation.*, in IEEE Symposium on Computers and Communications (ISCC'13), ISCC '12, 2013.

[7] A. Celesti, M. Fazio, M. Villari, A. Puliafito, and D. Mulfari, *Remote and deep attestations to mitigate threats in cloud mash-up services*, in World Congress on Computer and Information Technologies (WCCIT'13), Sousse, Tunisia, 2013.

[8] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, *Integration of clever clouds with third party software systems through a rest web service interface*, in IEEE Symposium on Computers and Communications (ISCC'12), ISCC '12, 2012, pp. 827–832.

[9] D. Diaz-Sanchez, F. Almenarez, A. Marin, D. Proserpio, and P. Arias Cabarcos, *Media Cloud: an open cloud computing middleware for content management*, IEEE Transactions on Consumer Electronics, 57 (2011), pp. 970–978.

[10] M. Fazio, M. Paone, A. Puliafito, and M. Villari, *Huge amount of heterogeneous sensed data needs the cloud*, in SSD'12, 2012.

[11] H. Flores and S. N. Srirama, *Dynamic Re-configuration of Mobile Cloud Middleware based on Traffic*, in IEEE MASS'12), October 8-1 2012.

[12] A. Juels and A. Oprea, *New approaches to security and availability for cloud data*, Communication of the ACM, 56 (2013), pp. 64–73.

[13] P. Leitner, B. Satzger, W. Hummer, C. Inzinger, and S. Dustdar, *Cloudscale: a novel middleware for building transparently scaling cloud applications*, in SAC'12, 2012, pp. 434–440.

[14] E. Manias and F. Baude, *A component-based middleware for hybrid grid/cloud computing platforms*, Concurrency and Computation: Practice and Experience, 24 (2012), pp. 1461–1477.

[15] H. Nagakura and A. Sakurai, *Middleware for creating private clouds*, Fujitsu Scientific & Technical Journal (FSTJ), 47 (2011), pp. 263–269.

[16] J. Povedano-Molina, J. M. Lopez-Vega, J. M. Lopez-Soler, A. Corradi, and L. Foschini, *Dargos: A highly adaptable and scalable monitoring architecture for multi-tenant clouds*, Future Generation Computer Systems, May (2013).

[17] A. Ranabahu and M. Maximilien, *A Best Practice Model for Cloud Middleware Systems*, in Best Practices in Cloud Computing: Designing for the Cloud, 2009.

[18] F. Valente, G. Zacheo, P. Losito, and P. Camarda, *A telecommunications framework for real-time monitoring of dangerous goods transport*, in Intelligent Transport Systems Telecommunications,(ITST),2009 9th International Conference on, October 2009, pp. 13 –18.

[19] Z. Wenbing, P. Melliar-Smith, and L. Moser, *Fault Tolerance Middleware for Cloud Computing*, in IEEE 3rd CLOUD'10, July 2010, pp. 67–74.

[20] Z. Yingjun, X. Shengwei, X. Peng, and W. Xinquan, *Shipping containers of dangerous goods condition monitoring system based on wireless sensor network*, in Networked Computing (INC), 2010 6th International Conference on, may 2010, pp. 1 –3.

Edited by: Maria Fazio and Nik Bessis
Received: Nov 2, 2013
Accepted: Jan 10, 2014

DOI 10.12694/scpe.v14i4.929
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. 215–233. http://www.scpe.org

MOBILES FOR SENSING CLOUDS: THE SAAAS4MOBILE EXPERIENCE *

SALVATORE DISTEFANO† GIOVANNI MERLINO ‡ AND ANTONIO PULIAFITO§

Abstract. Smart devices, and mobiles in particular, are at the forefront of several hot new trends in ICT, such as the Internet of Things and service computing. Cloud computing is another paradigm generating a great deal of offshoots, some of which are aimed at enabling novel services and applications by exploiting its ubiquity and flexibility in combination with sensors and the (meta)data they produce about phenomena, events and other interesting items about the physical world. In this context, the authors, propose a new way to orchestrate devices, in particular SNs and mobiles, as resources to build up Clouds of sensors, reverting the current wisdom about mobile Clouds, i.e. the integration of feature-rich devices into the Cloud fabric as mere clients to one where personal / wearable devices are actively involved into a "sensing" Cloud, forming a fully feedback-enabled ecosystem. The main aim of the Sensing and Actuation as a Service (SAaaS) approach is therefore to implement such a Cloud by enrolling and aggregating sensing resources from sensor networks and personal, mobile devices. A device-centric approach is embraced as in IaaS Clouds: once collected, the physical (sensing) resources are abstracted and virtualised and then provided elastically, on-demand, as a service to end users, including facilities for customization of the (hosting) embedded platform.

A key point of the SAaaS approach is the abstraction of resources, i.e. providing a uniform way to access to and interact with the underlying physical nodes. In this paper we focus on the low-level interaction with sensing resources in SAaaS, restricting the scope to mobiles, thus providing details on theoretical and design aspects as well as technical and implementation ones. In particular, we report on an implementation of the SAaaS low-level modules on Android devices, the SAaaS4Mobile one, providing architectural descriptions of the main modules, implementation guidelines and discussing through a preliminary implementation evaluation the effectiveness of the approach.

Key words: Cloud, sensors and actuators, sensing abstraction and virtualisation, OGC Sensor Web Enablement, Android.

1. Introduction and Motivations. Since their introduction and adoption, in early '90s, mobiles strongly impacted on everyday life changing, sometimes improving, the lifestyle. This transformation process is based on the advancements of network and processor technologies, following new trends and forms of interactions. Current smartphones have computing, storage and sensing capacities that can be compared to laptop or desktop, as a new, radical, interpretation of personal computing, the personal device or PDA frontier. This state of affairs has unlocked new ideas and paved the way for rethinking and reinterpreting foundational technologies such as Internet, driving efforts towards the Internet of Things (IoT), or service engineering at the basis of the Cloud computing and the Web 2.0.

At a lower scale, this also inspired us in developing an idea mixing aspects of both IoT and Cloud, involving any form of sensing resource such as sensor networks as well as standalone smart devices into a wide-area (geographic) sensing infrastructure. Our idea was to gather and collect sensing and actuation resources from contributing nodes, either sensor networks or personal, mobile devices, following a volunteer-based approach to build up a scalable sensing infrastructure, on-demand providing sensing resources to end users according to their requirements, as computing resources in Infrastructure as a Service (IaaS) Clouds, towards sensing Clouds. Thus, in previous work [9, 10] we proposed the Sensing and Actuation as a Service (SAaaS) framework to deal with the issues arising in sensing Clouds, mainly referring to resource abstraction and virtualisation, enrolment, indexing, discovery and management.

According to the SAaaS approach, the sensing Cloud provider that builds up and manages the infrastructure, has to provide actual sensing and actuation resources, even if abstracted and virtualised through a specific framework, to the user. Users may therefore handle, manage and customize (virtual) sensing resources at their will, according to their needs, for example for inclusion into an existing sensor network they administrate, e.g. when not able anymore to guarantee coverage of a certain area. This way the user resorts to a sensing Cloud

*This work is partially funded by PhD programme under grant PON R&C 2007/2013 "Smart Cities" and by Simone project under grant POR FESR Sicilia 2007/2013 n. 179. The authors would also like to acknowledge networking support by the COST Actions IC1203 - ENERGIC and IC1303 - AAPELE.

†Dip. di Elettronica, Informazione e Bioingegneria Politecnico di Milano Piazza L. Da Vinci 32, 20133 Milano, Italy. Email: salvatore.distefano@polimi.it

‡Dip. di Ingegneria, Università di Messina, Contrada di Dio, 98166 Messina, Italy. Email: gmerlino@unime.it

Dip. di Ingegneria Elettrica, Elettronica e Informatica, Università di Catania, Viale Andrea Doria 6, 98166 Catania, Italy. Email: giovanni.merlino@dieei.unict.it

§ Dip. di Ingegneria, Università di Messina, Contrada di Dio, 98166 Messina, Italy. Email: apuliafito@unime.it

once the resources are not enough, asking for further sensing resources, in a Cloudbursting fashion. We defined our approach as *device-centric* against the *data-centric* one, which aims at providing the user just with, more or less elaborated, sensed data, hosted by traditional Cloud providers [12], thus a Software as a Service (SaaS) approach at the core. A really key difference between the two approaches is that in the device-centric one the user obtains actual sensing resources that could be configured and managed according to user requirements, while in the data-centric one the user can just retrieve data gathered and offered by the provider itself, since the sensing resource may not be handled directly by the former, because it is managed by the provider itself.

One of the key features of a device-centric approach is the abstraction of sensing resources, i.e. techniques for abstracting details and mechanisms from heterogeneous hardware solutions, to access, interact and communicate with the sensing resources. In the SAaaS paradigm such functionalities are delegated to the Hypervisor component, which also provides virtualisation mechanisms. This kind of low-level albeit mediated (shared-)access to resources is what bears most of the challenges to be investigated to design and implement an IaaS-like Cloud such as the SAaaS one.

In this paper we specifically focus on the low-level interaction with sensing resources belonging to SAaaS-enabled infrastructure, as a first step for a bottom-up implementation of the framework. Specifically, we start our investigation by considering smart devices only, thus developing a first prototype of the SAaaS abstraction layers for mobiles, the *SAaaS4Mobile*.

When mobiles are involved, certain advantages of our device-centric approach are more easily recognizable, e.g. opening up opportunities for direct involvement of (i.e. notifications to) the device owner / resource contributor, a chief requirement for mobile crowdsensing scenarios. On the other hand, for all its worth, enrolment of mobiles bears many unique challenges, to be tackled by means of volunteering approaches, in order to satisfy certain predefined (e.g. SLA-mandated) reliability and QoS requirements [8, 11]. Moreover, reward systems are one of the approaches to be called for mobiles, as very peculiar motivations play their role when dealing with volunteering owners of resources with such hard constraints (e.g. relatively fast battery depletion [4]).

We therefore characterize the overall Hypervisor architecture [10] for mobiles, since some of the components previously identified as distinct modules in the general case, i.e. also including SNs, merge when dealing with standalone smart devices. Then, we focus on the main, mandatory components of the SAaaS4Mobile Hypervisor architecture, investigating and implementing a solution with specific regards to the abstraction layers modules. The implementation of SAaaS4Mobile abstraction modules has been tackled for Android mobiles only, at this stage, starting from well-known standards as the Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE) [23] ones. To demonstrate the feasibility of the SAaaS approach the SAaaS4Mobile core modules implementation has been tested through a proof of concept in which the performance of the main low level operations have been measured. The values thus obtained fosters further developments of the paradigm, since they provide useful insight on its effectiveness.

In order to explain in detail our work we organized the paper as follows. Section 2 provides an overview of the state of the art on related work, mainly focusing on some specific aspects provided as background. The Hypervisor architecture is discussed in cf. Sect. 3, with particular emphasis on the standalone device characterization into SAaaS4Mobile. The corresponding architecture for abstraction layer modules is then discussed in cf. Sect. 4, also describing possible interactions with the system from a dynamic-behavioural perspective in cf. Sect. 5. Details on the implementation of a preliminary version of the SAaaS4Mobile stack on Android and SWE standard environments are reported in cf. Sect. 6 with the evaluation of a proof of concepts. Finally, cf. Sect. 7 summarizes the paper providing hints for future work.

2. Preliminary Concepts.

2.1. Related Work. In sensing environments, software abstraction layers allow to address interoperability and communication issues [21, 27] to enable the dynamic reconfiguration of sensor nodes [19, 2], and to manipulate sensor data [1, 20].

Some solutions are specifically conceived for building up networks of mobiles' sensors. The Mobile phone Sensor Network [6] allows to collect observations from Bluetooth-enabled sensors on mobiles and send them to a database through an OGC SWE SOS. Similarly, [18] allows to perform the injected measurements and to express them in SWE-compliant format, also resorting on the mobile computing and storage resources.

SWE [23] is a standards' suite for achieving abstraction and interoperability in sensor networks, widely used [6, 18, 5] in the implementation of sensing Web services as well as to address abstraction issues in *virtual sensor networks* [24]. It comprises several standards such as the *Sensor Model Language* for the description of sensor systems and processes associated with sensor observations; the *Observations & Measurements* to express observations and measurements through standard Web service interfaces; the *Sensor Observations Service* (SOS) to collect observations and system information; the *Sensor Planning Service* (SPS) to plan observations; the *Sensor Alert Service* (SAS) to publish and subscribe sensor alerts; the *Web Notification Service* (WNS) to asynchronously deliver messages and alerts from SAS and SPS.

In [27], a framework to enable management of physical sensors on IT infrastructure, abstracting and virtualising them into virtual sensors is provided. In [25] an infrastructure for connecting sensor networks and applications is proposed, allowing to select physical sensors in wide-geographic sensor networks that can be implemented as suggested in [13].

A Semantic Web Architecture for Sensor Networks (SWASN) is proposed in [17], specifically dealing with inference on the sensor data collected by several heterogeneous SNs. In [5] a service-oriented framework to integrate heterogeneous sensors and virtual ones is described, starting from the SWE SPS, WNS, and SOS.

In [3] Android mobile sensors are categorized into two different classes depending on the functionality provided (common an complimentary) and are considered as potential providers of raw data. To recognize user context information the mobile phone infrastructure of [14] can be used, e.g. for monitoring physical actions performed by users such as walking and running. Similarly, [26] allows to recognize physical activities of mobile phone owners-users.

2.2. Background. In this section we explore the topics about the background of SAaaS4Mobile: a high-level description for the SAaaS paradigm, including a depiction of its layering, followed by a subsection on standards relevant to our effort, in this case SWE by OGC, and by details on the specific platform to be addressed, i.e. Android.

2.2.1. SAaaS. The main aim of SAaaS [9] is to adapt the IaaS paradigm to sensing platforms, bringing to a Cloud of sensors, where sensing and actuation resources may be discovered, aggregated, and provided as a service, according to a Cloud provisioning model.

The inclusion of sensors and actuators in geographic networks as Cloud-provisioned resources brings new opportunities with regards to contextualization and geo-awareness. By also considering mobiles, possibly joining and leaving at any time, the result can be a highly dynamic environment. The issue of node churn can only be addressed through volunteer contribution paradigms [9, 7]. Furthermore, the SAaaS has to manage contributions coming from sensor networks, mobiles or any other "smart" device equipped with sensors and actuators, to ensure interoperability in a Cloud environment. It must also be able to provide the mechanisms necessary for self-management, configuration and adaptation of nodes, without forgetting to provide the functions and interfaces for the activation and management of voluntarily shared resources.

The SAaaS reference architecture [9] comprises three modules, Hypervisor, Autonomic Enforcer and VolunteerCloud Manager, shown in cf. Fig. 2.1. The *Hypervisor* allows to manage, abstract, virtualise and customise sensing and actuation resources that could be provided by enrolling either mobile device or SN *nodes*. Among key features are: abstraction of devices and capabilities, virtualization of abstracted resources, communications and networking, customization, isolation, semantic labeling, and thing-enabled services. All these features are presented in the next section. At a higher level with respect to the Hypervisor, the *Autonomic Enforcer* and the *VolunteerCloud Manager* deal with issues related to the interaction among nodes. The former is responsible of the enforcement of local and global Cloud policies, subscription management, cooperation on overlay instantiation. The VolunteerCloud Manager is in charge of exposing the Cloud of sensors via Web service interfaces, indexing of resources, monitoring Quality of Service (QoS) metrics and adherence to Service Level Agreements (SLAs).

2.2.2. OGC: Sensor Web Enablement. The OGC provides a large number of specifications, among which we can find the Sensor Web Enablement (SWE) family of standards. Designed for the management of sensor data on the web, as mentioned in [23], a unique framework of open standards for exploiting Web-connected sensors and sensor systems of all types is the focus of the specifications.

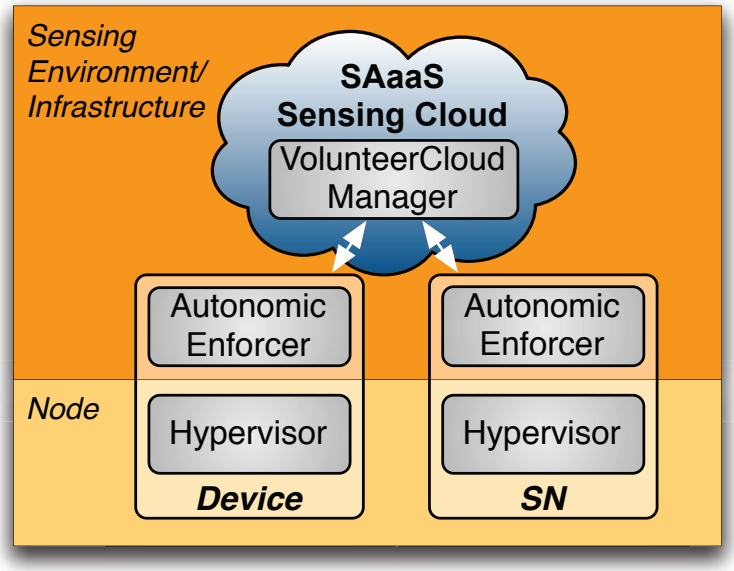


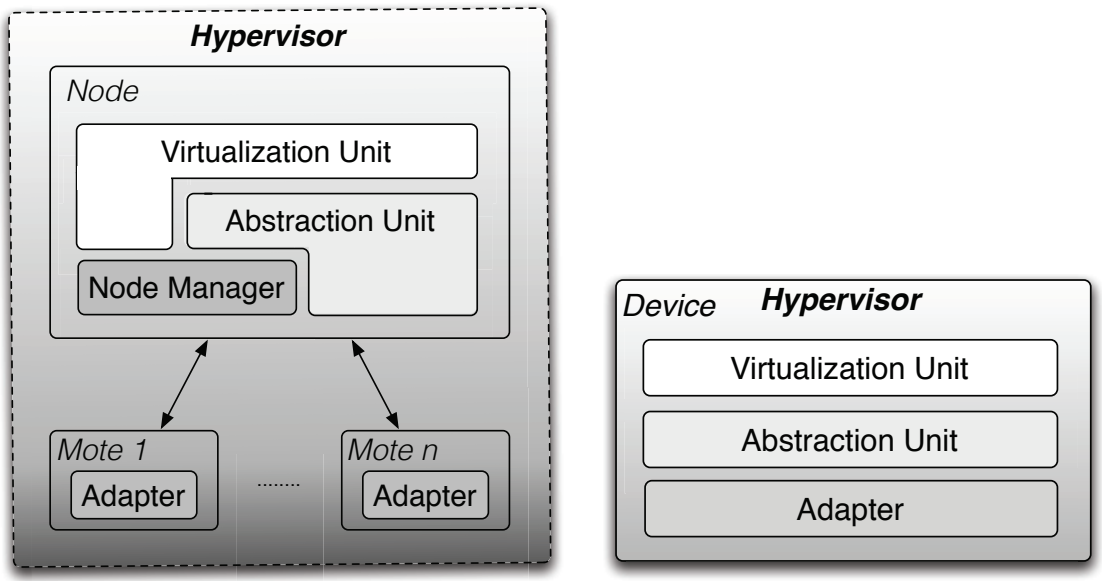
FIG. 2.1. *SAaaS reference architecture*

SWE standards aim at making all types of Web sensors, instruments, probes, and imaging devices accessible and controllable on the Web. The SWE framework is composed of seven standards, four of them have been approved as official standards by the OGC members.

- SensorML: it is a language based on XML schema to describe the sensor systems. It encodes a lot of features for sensors, such as discovery, geolocation processing observations, mechanisms for sensor programming, subscriptions to sensor alerts. In particular, it provides standard models and XML schemas to describe processes, and instructions for obtaining information from observations. SensorML enables discovery, access and query execution for the processes and sensors it models.
- Observations & Measurements (O & M): this model in particular is featured in the SOS specification, coupled with an XML encoding for observations and measurements originating from sensors, and archived in real-time. It provides standardized methods for accessing and exchanging observations, alleviating the need to support a wide range of sensor-specific and community specific data formats.
- Sensor Observation Service (SOS): it corresponds to the Observation Agent specified in the previous section. This is the service responsible of the transmission of measured observations, from sensors to a client, in a standard way that is consistent for all sensor systems including remote, in-situ, fixed and mobile sensors. It allows the customer to control the measurement retrieval process. This is the intermediary between a client and an observation or near real-time sensor repository.
- Sensor Planning Service (SPS): it corresponds to the Planning Agent, whose design and implementation are also addressed in this paper.

2.2.3. Mobile OS and IDE. Android is a mobile operating system developed by Google, in reality a software platform composed of five parts [16, 28].

- Applications: the platform provides a set of core applications, which includes email client, SMS app, contacts app, calendar, mapping, Web browser, etc.
- Application Framework: the base framework for developing Applications in Android. It is composed of a set of tools enabling the realization of applications. Moreover, security implications and privacy protection, among core concerns about such a device-driven approach, are mostly taken care by the Application Framework itself, deviating most of the attention the topic, here unaddressed accordingly, would otherwise deserve, e.g. in mixed / WSN-powered topologies.
- C/C++ Library: Android includes a set of C/C++ libraries, used by various components of the Android



(a) SN (b) Device

FIG. 3.1. *The Hypervisor modular architecture.*

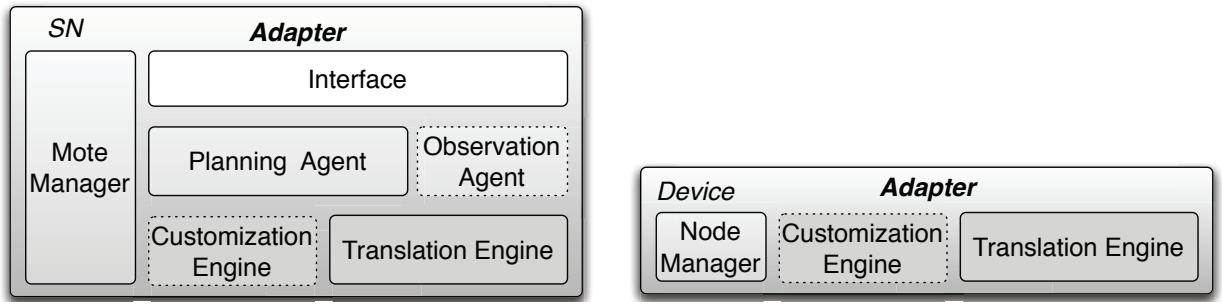
system. These capabilities are exposed through the Application framework.

- Android Runtime Library: It provides a big part of functionalities available in the core libraries of the Java programming language.
- The Linux kernel: Android relies on a Linux 2.6-based kernel for core system services such as memory management, process management, network stack, security, driver model, etc. The core also acts as a abstraction layer between the applications and the hardware.

In order to develop an application, Android offers a Software Development Kit (SDK) and a Native Development Kit (NDK) [15]. The SDK provides a large number of development tools, needed to build, test, and debug Android applications, including an emulator, able to emulate an Android mobile.

On the other hand, the NDK is the Android operating system API to natively develop in the language of the target architecture, as opposed to the Android SDK, which provides (Java) bytecode-based abstractions, thus hardware independent ones. The NDK helps developers integrate (typically, ARM) native code in their applications to exploit the performance offered by accelerated operations in the processor. Based on the Dalvik virtual machine, developers can embed C or C++ code to reuse some classes that have already been developed for other systems.

3. The Hypervisor. A *Hypervisor* [10] can be viewed as the foundational component of our device-driven approach to infrastructure-focused Clouds of sensors: it manages the resources related to sensing and actuation, introducing layers of abstraction and mechanisms for virtualization. It works at the node level, which could be either a whole sensor network, under a unique administrative domain, or a set of sensors, as built-in to a standalone device. In other words, also referring to cf. Fig. 2.1, a *node* could be either a whole sensor network, composed of several sensor boards or *motes* managed by a sink and/or a gateway, or a personal device that could be more or less smart and thus can be equipped with one or more sensors. The Hypervisor functionalities should fill this gap, dealing with such heterogeneity, hiding it to the above modules of the SAaaS reference architecture. In the SN case, it is therefore necessary to split the Hypervisor architecture between the SN node and mote layers. This way, all the motes composing an SN should have installed a specific Hypervisor module locally managing the motes, coordinated by the high level modules of the Hypervisor deployed on the node/SN gateway. This kind of two-level separation of concerns and assignment of operations descends also from the



(a) SN (b) Device

Fig. 3.2. *Adapter modules.*

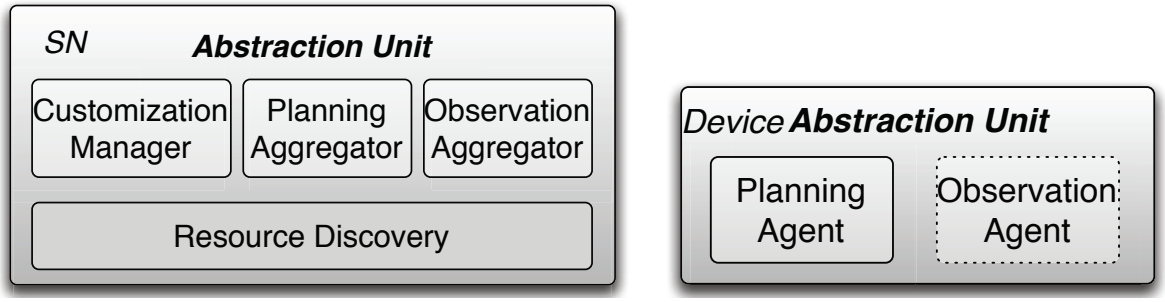
need for certain duties to be (self-)managed through autonomic approaches, typical of distributed entities.

A high-level, modular view of the Hypervisor architecture comprises four main building blocks, when dealing with SNs: Adapter, Node Manager, Abstraction Unit and Virtualization Unit as shown in cf. Fig. 3.1(a), collapsing down to three out of four components, when standalone devices (e.g. mobiles) are involved, as depicted in cf. Fig. 3.1(b).

At the bottom of the Hypervisor, there is the *Adapter*, which plays several distinct roles through its modules. The *Translation Engine* is a platform-specific driver, in charge of converting the high-level directives in native commands. The Hypervisor is also appointed for processing requests for reconfiguration of the device, using the (optional) *Customization Engine*, an interpreter able to execute on the sensing device the code needed to tailor the sensing activities to customer-mandated requirements. Yet, the most important duty this layer-spanning module has to cope with consists in providing mechanisms for the customer to establish an out-of-band (i.e. not Interface- or Agent-mediated) channel to the system, for direct interaction with either the resources (e.g. for Agent-agnostic collection of observations) or low-level modules (e.g. the Customization Engine), thus pinning it as a mandatory component of the architecture.

The *Node Manager* works only at the node level and is in charge of the basic sensing resource operations and mandating policies, in cooperation with the Adapter *Mote Manager* that replicates its functionalities at mote level. In standalone device the two modules are collapsed into the Adapter Node Manager (cf. Fig. 3.2(b)). It is important to remark that the depiction (cf. Fig. 3.1(a)) of the Virtualization Unit for SNs is L-shaped because it can work directly over the Agent-hosting Adapter in selected cases, e.g. when dealing with a degenerate SN made up of a single mote. In such cases, an autonomic approach is adopted delegating some management tasks of the Adapter to the Mote Manager running on the mote-side, performing specific operations such as power-driven self-optimization with the Node Manager. The Mote Manager is not needed on a standalone device, where the Node Manager itself makes up for the combination of the two modules.

The *Observation Agent*, featured in the Adapter for SNs (cf. Fig. 3.2(a)) and in the Abstraction Unit for mobiles (cf. Fig. 3.3(b)), is in charge of requesting, retrieving, and eventually pre-processing measurements. The *Planning Agent* (PA) pushes requests for actions (*tasks*) to the device. It is featured in the Adapter or the Abstraction Unit, depending on the kind of topology, as also for the Observation Agent. The requests leaving the PA are for preparing the resource to carry out a variety of duties (reservation of functionalities, tuning of parameters, scheduling of observations). These commands allow management of operating parameters such as duty cycle, sampling frequency, etc. Although providing useful and standardized mechanisms, the Observation Agent is not strictly needed to let customers exploit sensing infrastructure. Indeed the PA is enough to handle the physical (or virtualized) resources as long as a working bidirectional communication channel is established between the client and the mote or mobile hosting the sensing device. Such a facility would then be enough to let the customer do what is needed for getting and storing observations e.g. even build a client-side version of a SensorML-compliant module for management of observations, synchronously working over the channel if required.



(a) SN (b) Device

FIG. 3.3. *Abstraction unit modules.*

Furthermore, in SN deployments, the Adapter has to expose a standard-compliant customer-friendly *Interface* to on-board resources.

The Adapter provides its functionalities to the upper level *Abstraction Unit*. As can be seen comparing cf. Fig. 3.3(a) with cf. Fig. 3.2(a), with regards to SNs it replicates planning and observation facilities, modeled after those featured in the Adapter, but on a node-wide scale, combining the pool of resources of the whole SN. In particular, the *Observation Aggregator* exposes all resources from the nodes and the *Planning Aggregator* manages this set, sending combos, i.e. combination of commands, and tracking exit codes, eventually reacting to (partial) failures by triggering apt adjustments. The *Resource Discovery* module, which offers an interface to the motes, actively gathering descriptions of underlying resources and forwarding the results to the Aggregator modules. The *Customization Manager* acts as an orchestrator for customization engines located on the motes.

With regards to mobiles, the architecture of the Abstraction Unit degenerates from the one depicted in cf. Fig. 3.3(a) to the one in cf. Fig. 3.3(b), where only planning- and observation-oriented modules are part of the unit, now named Agents for taking on the same role as their counterparts in the Adapter for SNs (cf. Fig. 3.2(a)), thus leaving only the lowest layer of the latter in the version of the Adapter geared towards standalone mobiles (cf. Fig. 3.2(b)).

All components of the Abstraction Unit in cf. Fig. 3.3(a) are mandatory (solid line border), as those are needed to coordinate operations of the corresponding mote-side modules (when present), while the Observation Agent in cf. Fig. 3.3(b) is optional (dashed line border), as its counterpart is in cf. Fig. 3.2(a).

The highest level of the Hypervisor is the *Virtualization Unit*, name after the Virtual Machine Monitor to highlight its role as a manager of the lifecycle of virtualized resources instance. This includes APIs and functionalities for virtual instance creation, reaping and repurposing, as well as for boot- (defined statically) and run-time (dynamically) parameters discovery and tuning in accordance to contextualization requests.

4. Basic Abstraction Modules. Aim of this section is to provide details on the main modules implementing abstraction and adaptation facilities of the SAaaS4Mobile framework on mobiles.

4.1. Translation Engine and Node Manager. The SAaaS4Mobile Node Manager of cf. Fig. 3.2(b) is mandatory exclusively for its role as an out-of-band conduit, e.g. to the Customization Engine, where otherwise the only way to interact with sensing resources is mediated by the Planning Agent, leaving no path to mould the platform itself to the sensing requirements at hand. A way to establish this channel then, under a mobile platform with typical features such as Android, can be through platform-provided facilities, i.e. Google Cloud Messaging.

With regards to the Translation Engine, any sensing-related APIs provided by the node should be used when available since developing the Agents against those frees us from the need to implement a layer for command translation. For example, Android provides platform-specific Sensing APIs. Yet we cannot fully dispose of the Translation Engine, because not all onboard devices, which could be leveraged for sensing, are exposed as such (i.e. under the sensing APIs). This way the Translation Engine exploits the platform-standard (e.g. Android

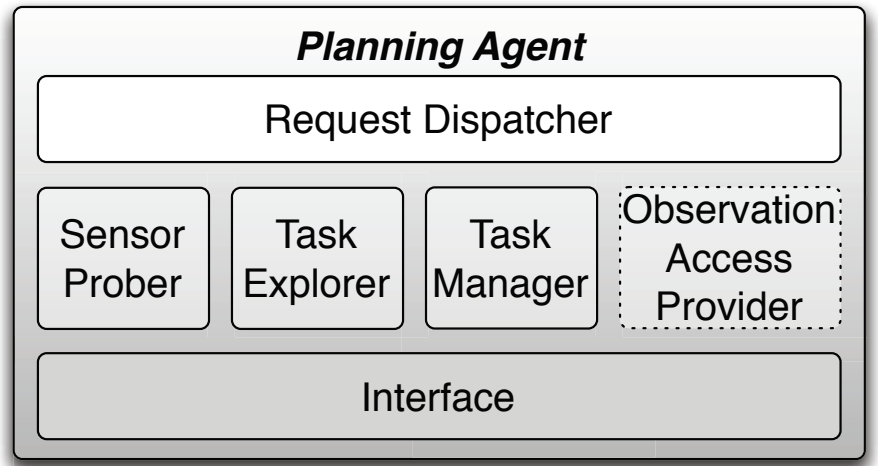


FIG. 4.1. *Planning Agent architecture*

and Linux) low-level tools and mechanisms in order to export these resources under an extended set of devices the sensing APIs know about, easing the job of the Sensor Prober, e.g. just letting it enumerate resources through the sensing APIs.

4.2. The Planning Agent. The SAaaS4Mobile Abstraction Unit, as shown in cf. Fig. 3.3(b), is mainly composed of the Planning Agent (PA) that is the only mandatory module of the unit, whereas the same building block resides in the Adapter when considering SNs. In the following, the details about the PA and its components are to be considered valid in both cases. The PA works side-by-side with the Observation Agent, complementing its features. Unlike the latter, engaged in providing upper layers with XML-encoded measurements (*observations*), sampled while driving the sensing resources, the former is mainly devoted to tune sampling parameters according to user-defined preferences, still to be interfaced with by means of extensible standards-compliant encoding of requests for *tasks*, and corresponding responses. Other than tuning, tasks for scheduling of observations can be consumed by the PA: it may be following a predefined schedule, or upon the occurrence of a particular event, or simply a request from a client. The main aim of this effort is exposing all underlying knobs to make them available for customers to operate on transparently.

In order to meet the aforementioned requirements, an architecture comprising the six modules shown in cf. Fig. 4.1 has been designed: a Request Dispatcher, a Sensors Prober, a Task Explorer, a Task Manager, an Observation Access Provider and an Interface. The *Request Dispatcher* has to identify and demultiplex a request to the modules underneath. The *Interface* has to interact with the SAaaS4Mobile Adapter services, i.e. the Customization Engine, the Translation Engine and the Node Manager.

The *Sensor Prober* is in charge of enumerating all the sensors and actuators within a sensors platform, however rich and complex, by low-level platform-specific system probing. These sensors are then identified according to their types, supported observation facilities and sampling specs, overall (nominal) features and manufacturing details (brand, model, etc.).

The *Task Explorer* is responsible for enumeration of available tasks, to be provided by probing sensors as listed by the aforementioned Prober. In terms of tasks, those related to parameter tuning for sensing resources logically differ depending on the sensor type and technology, it is therefore possible to e.g. plan retrieval of temperature samples from a thermometer, once a certain threshold has been exceeded, change the relative position and the focal length of a camera, or simply schedule reading of sensor observations at fixed intervals, etc. Moreover, in order to assess feasibility of a certain task, among the ones enumerated for selection, the sensor has to be queried and provide (runtime) confirmation, or else denial, of availability for servicing (or reservation thereof). It's then up to the querying party to decide what to do after feasibility assessment for the task under consideration.

The *Task Manager* controls tasks' lifecycle, since feasibility assessment through reservation/submission stages, then following up, and acting upon, running task progress. Due to the number of, and dependencies for,

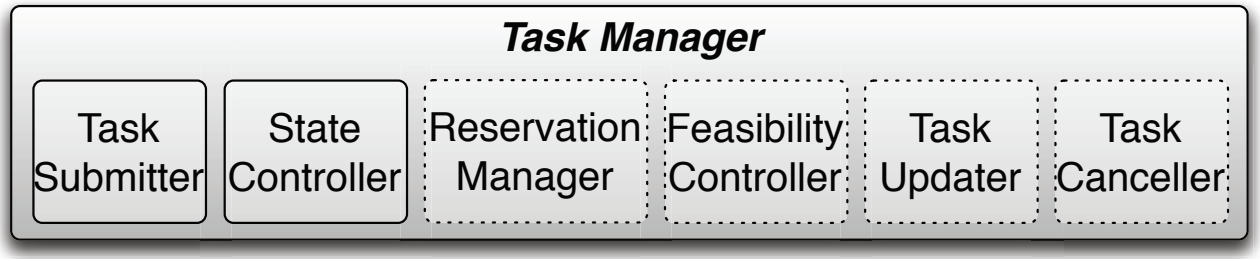


Fig. 4.2. *Task Manager architecture*

the operations involved, the Task Manager duties have been assigned to six modules as shown in the architecture of cf. Fig. 4.2. Two of them are mandatory, the rest are optional.

Task Submitter and *State Controller* implement mandatory functionalities. Their roles are, respectively, to enable users to set all (mandatory) parameters for a specific task before submission to a sensor, and submit it when ready, and to follow up the processing of the task, alerting any agent, subsequently querying about availability for task execution, about its (busy) status until completion. The optional modules are instead: Reservation Manager, Feasibility Controller, Task Updater and Task Canceller. These modules provide additional facilities for control on running (or yet to be scheduled) tasks to process.

If needed, a user may reserve a task for a period of time, during which he/she gets exclusive access to the underlying resource, as no other user can submit or reserve it. The task will then be executed as soon as the user confirms for the real processing stages to commence. The *Reservation Manager* is responsible for both reservation of tasks, and its confirmation. The *Feasibility Controller* has to check if a task is feasible, as detailed above. The feasibility of a task depends on the availability of any resource essential for task servicing, e.g. if not still allocated due to a previous request.

Then, the *Task Updater* is in charge of updating configuration parameters of a task, if some modifications have to be pushed after tasks enter into processing stages. Lastly, the *Task Canceller* empowers users to stop and therefore retire a task, when already submitted or under reservation.

Finally, once a task has been serviced, the resulting observation gets stored. Any observation will be accessible through the Observation Agent only. In terms of observations, the sole duty up to the PA lies in the *Observation Access Provider* ability to provide endpoints to access measurements. Being dependent on the Observation Agent, it is an optional component, required only if the latter is implemented in the Abstraction Unit.

5. A Dynamic Perspective. After having described the SAaaS4Mobile building block architecture, we can go into further details about the interaction among them from a dynamic, behavioural perspective. We identify three main phases of an end user-SAaaS4Mobile system interaction, as depicted in cf. Fig. 5.1:

i. *Sensors & Tasks Acquisition*: providing users with all available tasks, as offered by the SAaaS provider;
ii. *Sensor Use / Interaction*: selecting and preparing a task to be then submitted to the SAaaS provider, while keeping the ability to manage the task during its execution;
iii. *Observation Access*: in case one or more observations were the expected output of the task (e.g. scheduling or confirmed reservation), providing users with methods to retrieve stored measurements.

Resource release is not described here as it can be considered a special case degenerating from the management of requests for cancellation of tasks.

5.1. Sensors & Tasks Acquisition. The first macro-step of this workflow consists in the acquisition (enumeration) of all the tasks available over the full set of (on-board) sensing resources. More in detail, as depicted in cf. Fig. 5.2, an activity diagram (AD) for resource acquisition, a client sends a request, featuring requirements and preferences on the kind of needed sensing resources and corresponding range of tasks, to be submitted to the SAaaS4Mobile framework server exposed by the provider of choice. At this stage the high-level request gets mapped to standards-compliant constraints that can be easily be verifiable against enumerated resources and associated task types; the mechanisms for this mapping are out of the scope of this paper, probably the object of future investigation efforts. From the perspective of the contributor, e.g. mote-side, the job of the

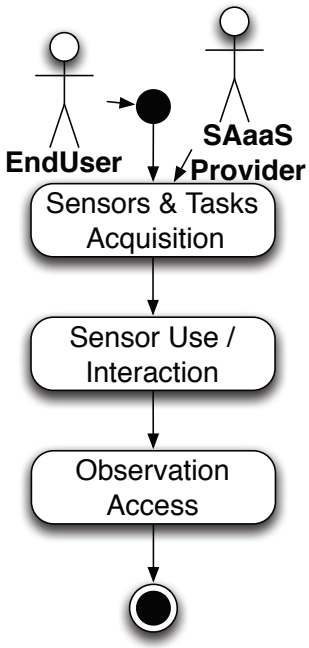


FIG. 5.1. *SAaaS4Mobile-end user interaction AD*

SAaaS4Mobile framework is to independently (i.e. at boot-up) probe the mote, at a level as close to hardware or OS / platform as possible, to find any exploitable sensing (or actuation) resource on board, e.g. not allocated exclusively to some other, immutable, activity. The probing thus happens at the very least in parallel, or possibly even long before the first request to be mapped comes in. The core resource (information) acquisition then happens by means of a two-step operation: the first being the search for capabilities, e.g. the kinds of phenomena the devices would sample during observations. The second one pursues the goal of retrieving all available tasks among the subset of sensing resources chosen by selection over advertised capabilities, i.e. according to one or more of the aforementioned criteria (e.g. phenomena to be sampled).

Again, upon reception of the list of available tasks, the SAaaS4Mobile framework server scans it to find a task matching the provided requirements. Moreover, the enumerated tasks provide a detailed description of parameters that can be set at the will of the customer. The list is sent as an endpoints' notification. If none correspondence was found, the SAaaS4Mobile framework server sends a notification to the client indicating that there is no results.

5.2. Sensor Use / Interaction. The second macro-step enables users to manage and configure a task, as obtained according to the first one. Therefore, the former can be split in its constituent macro-actions, and depicted accordingly in two ADs, the Submission (cf. Fig. 5.3) and Management (cf. Fig. 5.4) ones, respectively.

Submission operations, per the AD, comprise pre-submission configuration stages for a task, and submission itself. In the previous step of the high-level user-system interactions, the client has received a subset of available tasks, filtered by compatibility to constraints on capabilities and other requirements. At this point the client just has to choose one among the available alternatives for tasks, ready for reservation and submission, and finally configure it. At last, once the configuration is over, there are three different methods to submit the task, including configuration parameters, to the sensor:

- *direct submission* - a "Submit" request by the client gets forwarded to the sensor, being managed by the Task Submitter, while containing all the parameters needed to enable the resource to service the task under consideration.
- *submission by reservation* - reservation of a task for a beginning of processing stages in the future, under the guise of a "Reserve" request, which aims to book a resource (e.g. task) for a limited period

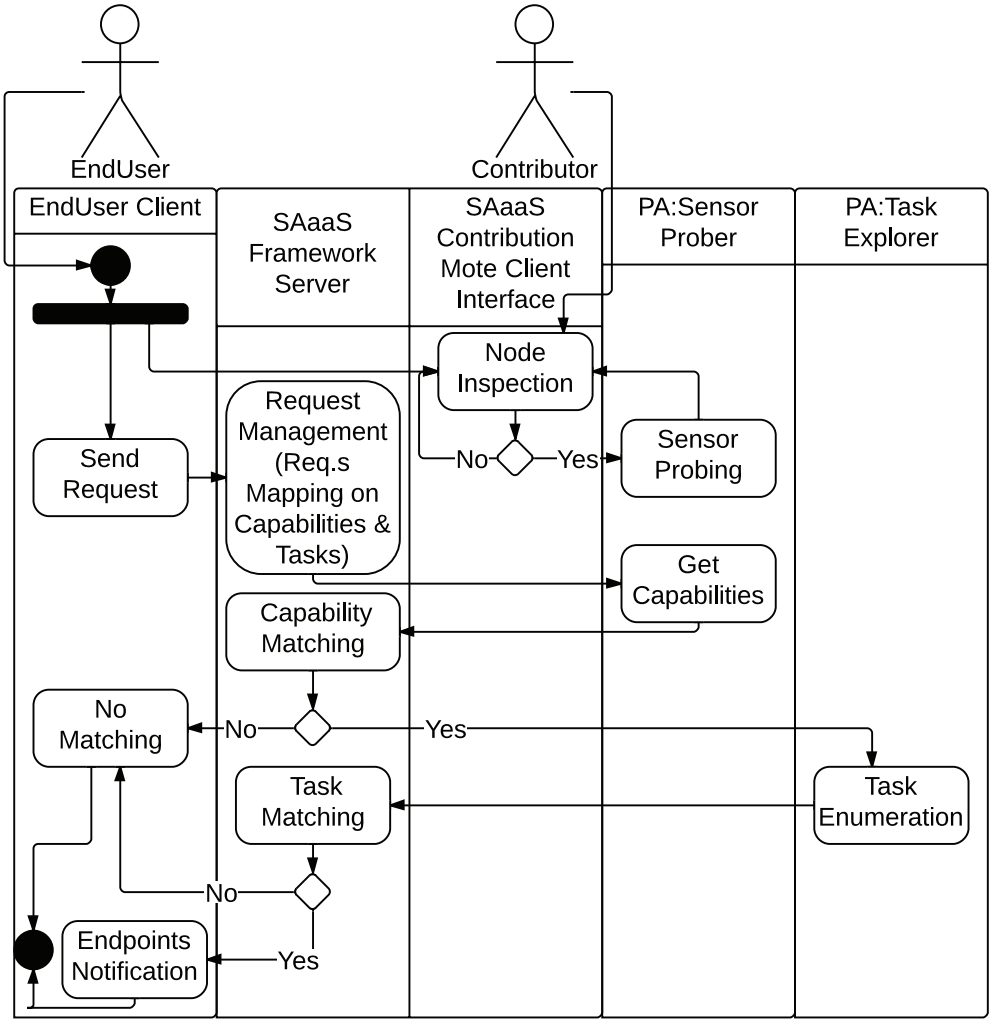


FIG. 5.2. *Resource acquisition AD*

under exclusive access. In case the resources are already allocated to another client or the configuration contains an error, no further progress can be achieved along the reservation attempt, apart from starting over. Otherwise, if the reservation was successful, a request to "Confirm" it may be sent, at the discretion of the client previously forwarding the reservation, upon which task processing commences, up to completion and subsequent deallocation of reserved resources, for the next request to be serviced. Both requests, i.e. reservation itself, and its confirmation, are managed by the Reservation Manager.

- *feasibility checking* - a "Get Feasibility" request, to be fully managed by the Feasibility Controller. The corresponding response signals approval or denial of subsequent submission / reservation operations, as evaluated at request time, thus dependent on availability of resources per conveyed requirements. If execution is evaluated as feasible, then the client can send a submit (or a reserve) request, and follow along one of the two aforementioned flows.

The AD related to Management describes the flows where the clients act upon an already running instance of a task execution, in particular empowered by three kinds of requests available at that stage:

- *status checking* -"Get Status" invocation to know at which step of the execution is the task;

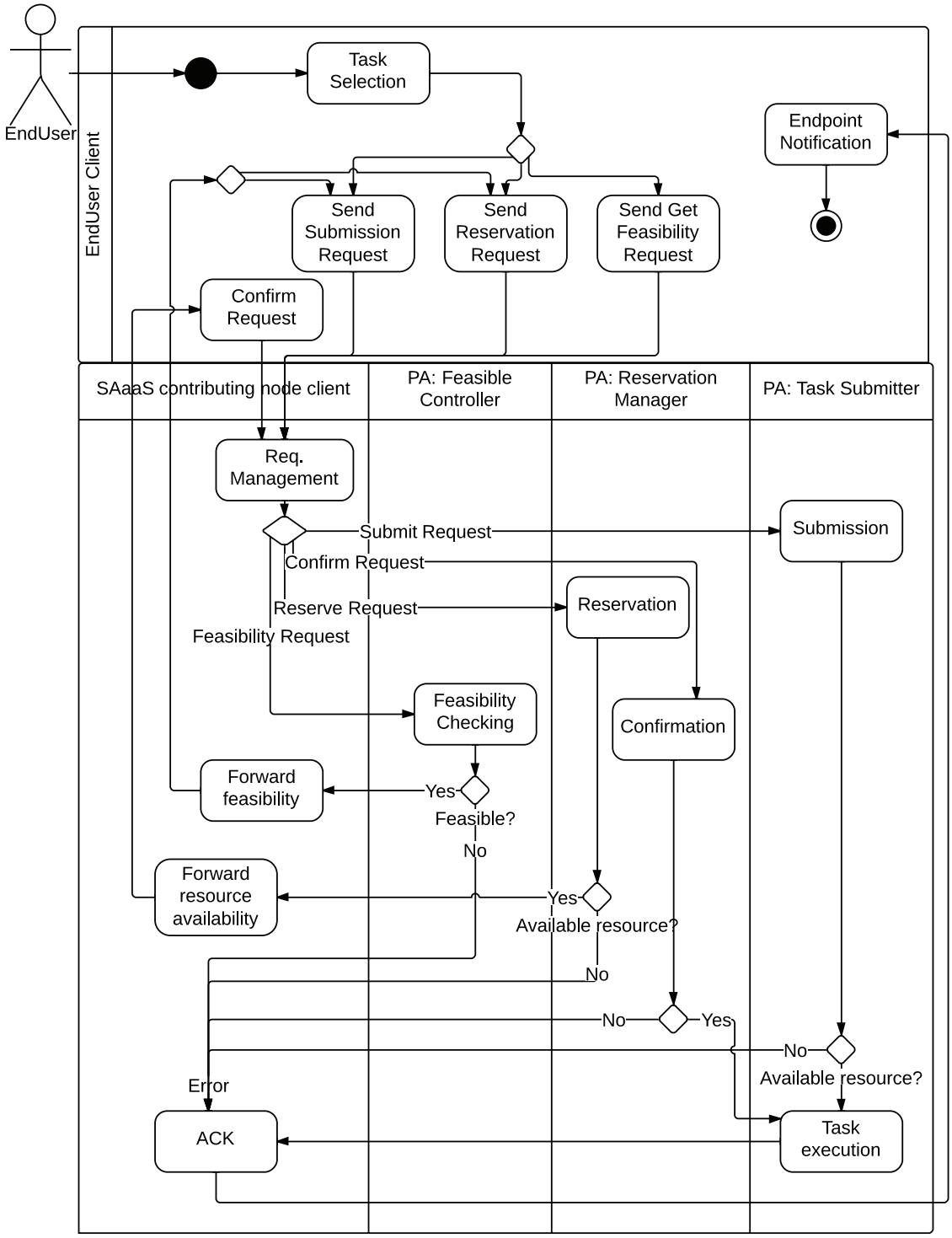


FIG. 5.3. *Interaction for submission operations*

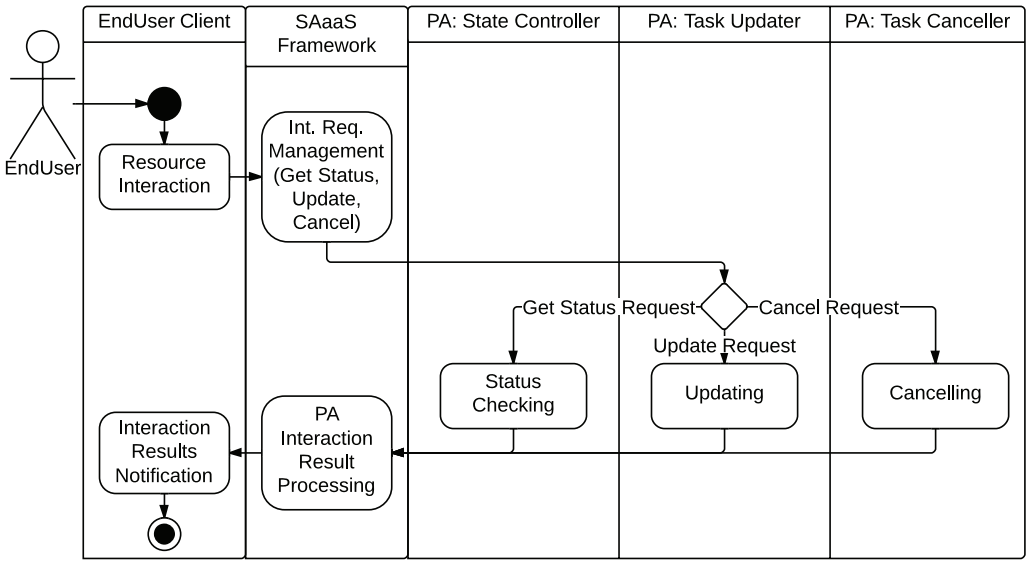


Fig. 5.4. *Interaction for management operations*

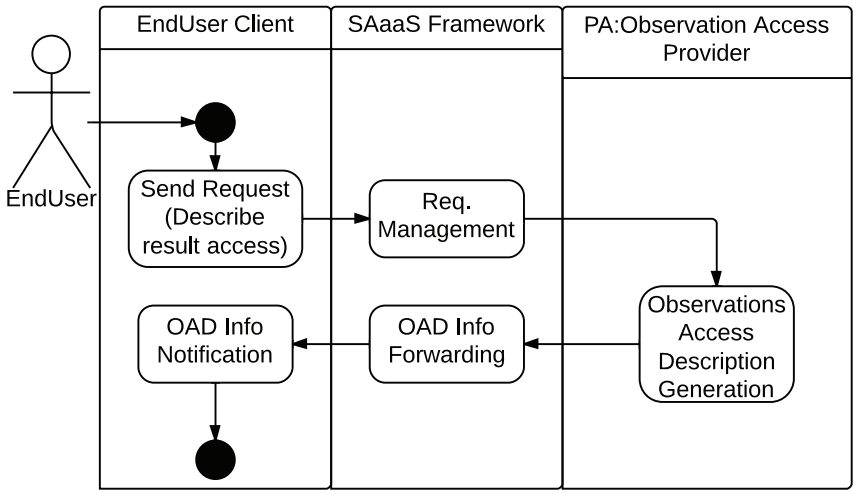


Fig. 5.5. *Observation access AD*

- *updating* - "Update" invocation to reconfigure task parameters;
- *cancelling* - a "Cancel" request to quit submitted or reserved tasks.

As can be seen by inspecting the swimlanes in the diagram of cf. Fig. 5.4, these requests are services respectively by: State Controller, Task Updater and Task Canceller, all components inside the Task Manager.

5.3. Observation Access. The last diagram, in cf. Fig. 5.5, depicts the flow for the user to get access to past observations, obtained thanks to the corresponding task(s). Indeed, as specified in cf. Sect. 4.2, the PA may also leverage Observation Agent services. So, after an interaction, when involving tasks to schedule observations, a client may later demand for endpoints and/or mechanisms to access data about obtained measurements, by means of "Describe Result Access" requests specifically. This translates to a transparent (to the user) interaction between the PA and the Observation Agent.

6. Proof of Concepts. In this section we detail on the implementation of a preliminary version of the SAaaS4Mobile framework including the very basic core abstraction modules and functionalities, which has been first described and then evaluated from an operational point of view.

6.1. Implementation. The implementation of the low-level modules of the SAaaS4Mobile framework has been targeted to mobiles equipped by Android OS 4.0, using the NDK developer libraries and API provided by the Android community [15]. The core of this effort, the design and coding of the Abstraction Unit, is based on the SWE Sensor Planning Service (SPS) 2.0 standard [22]. It enables the interaction among user clients and sensor and actuator services using XML schemas to submit requests and to allow the service to reply. Modeling behaviour after the SPS standard, the functionalities of the Sensor Prober, Task Explorer, Task Manager and Observation Access Provider modules described in cf. Sect. 4.2 have been developed.

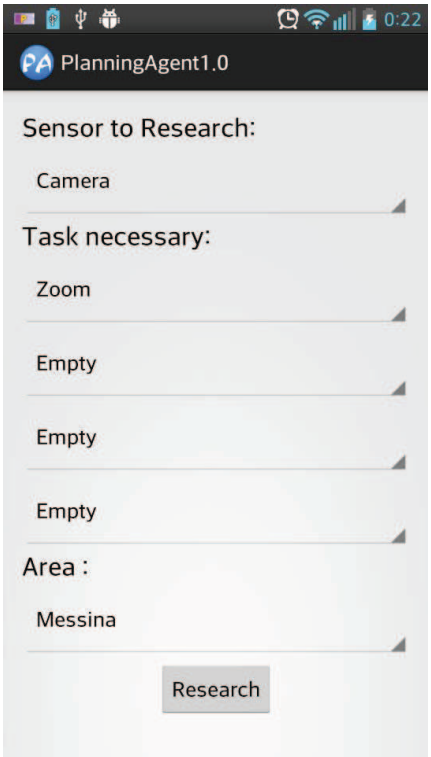
The Sensor Prober has to retrieve information regarding: i) the contributor, if available (the extent of such information disclosure is totally up to the contributor); ii) the node sensors and their descriptions, also including the measured phenomenon and corresponding metrics; and iii) the geographic area (range) inside which observations are significant. This feature is implemented by the SPS *GetCapabilities* primitive. A *GetCapabilities* request is composed of four sections. The first one is *ServiceIdentification* containing the contributing node metadata, i.e. generic info on the type of the node, brand, model and similar. Then the *ServiceProvider* section provides information on the contributor, if available and public. The third section is the *OperationsMetadata* one, with metadata about the operations specified by the service and implemented by the node. The last is the *Content* section, containing metadata about the sensors provided by the smart device through the PA and the communication mechanisms supported (XML, SOAP, etc.).

The Task Explorer retrieves the list of tasks that can be performed on a sensor through specific SPS *DescribeTasking* requests. A description of the available configuration operations for the sensor is thus obtained and provided to the Task Manager. As shown in cf. Fig. 6.1(a), the request just contains a *Procedure* element to enquiry a sensor in the list about the tasks that can be performed. The tasks are identified by the name, the description, and the capabilities' configuration information. The Task Manager implements a set of SPS requests. The *Submit* one allows the user to launch the execution of a configured task. Eventually, before to submit a job request, it is possible to enquire about its feasibility through the *GetFeasibility* primitive as shown in cf. Fig. 6.1(b). The reply, as depicted in cf. Fig. 6.1(c), can be "Feasible" or "Not Feasible" and, optionally, it may contain a list of alternative sets of tasking parameters that might help to the reformulation of a request. The user can also reserve the resources required to perform a specific task and then launch the task through the *Reserve* and *Confirm* requests as shown in cf. Fig. 6.2(a). In a *Reserve* request an expiration time has to be specified. At expiration time, all the reserved resource are released if the task has not been confirmed as in cf. Fig. 6.2(b).

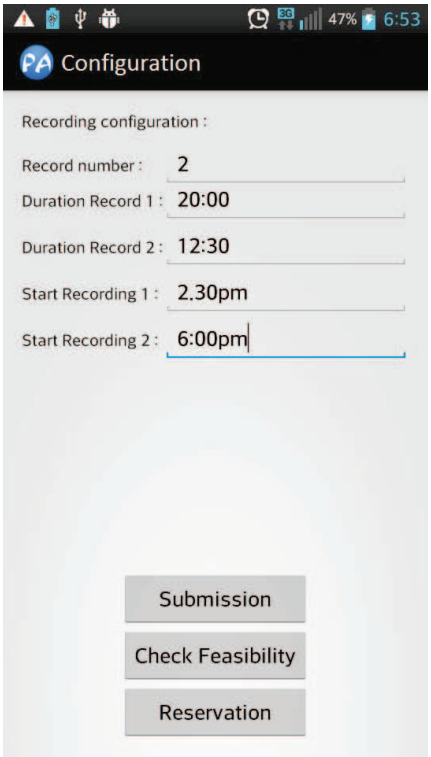
It is possible to check the status of a task using the *Status* request as shown in cf. Fig. 6.3(a). A task can be in six different states: "In Progress" if the service is executing it (cf. Fig. 6.3(b)), "Completed" if it was completed as planned, "Reserved" if it has been reserved, "Failed" if execution fails, "Expired" when the task reservation expires and "Cancelled" if the task was cancelled. The client can eventually update or cancel a task, with the *Update* and *Cancel* requests respectively.

Finally, the Observation Access Provider in the PA aims at providing the client with mechanisms, if needed, and endpoints to access the observations and measurements obtained during execution. It implements processing of SPS *DescribeTaskingResult* requests to interact with a specific sensor or a specific task as the ones shown in cf. Fig. 6.4.

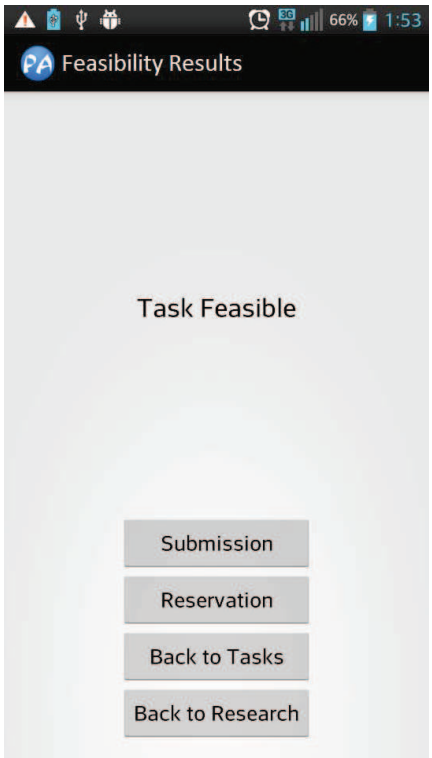
6.2. Preliminary Evaluation. In this section we provide some results on a preliminary prototype implementation of the SAaaS4Mobile abstraction layers. We have therefore implemented a mock SAaaS4Mobile testbed composed of an Intel I7 laptop acting as the SAaaS4Mobile server and an Android 4.2 Samsung S3 mobile as client. The SAaaS4Mobile client and server are implemented leveraging Java servlet technology using Apache Tomcat as the servlet container. The prototype is used to test the deployment and operation of very simple and basic operations as the ones discussed above. More specifically we evaluated *GetCapabilities*, *DescribeTasking*, *Submit* and *Observation Access* requests, by invoking them and iterating each test 1000 times, collecting the corresponding results to obtain the mean time and the standard deviation for each measurement as described in the following.



(a) Acquisition

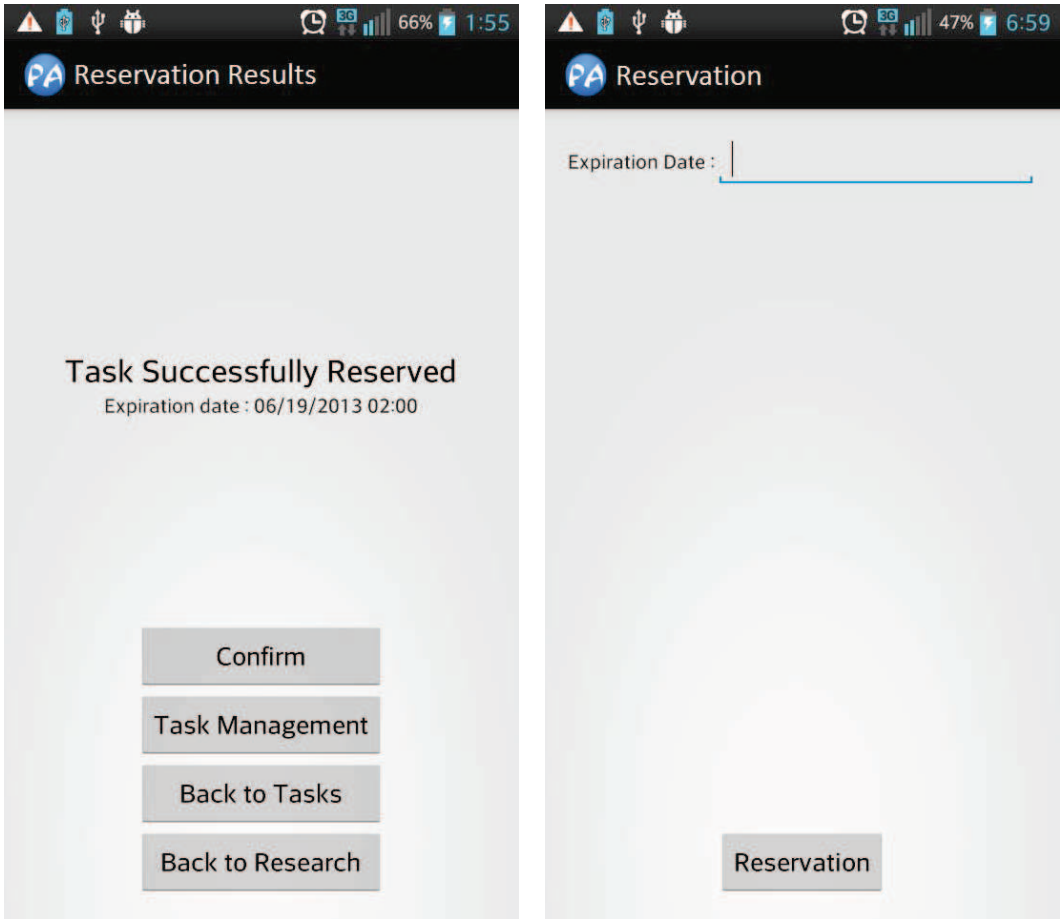


(b) Configuration



(c) Feasibility

FIG. 6.1. *SAaaS4Mobile resource acquisition, configuration and feasibility check*



(a) Reservation (b) Expiration timeout

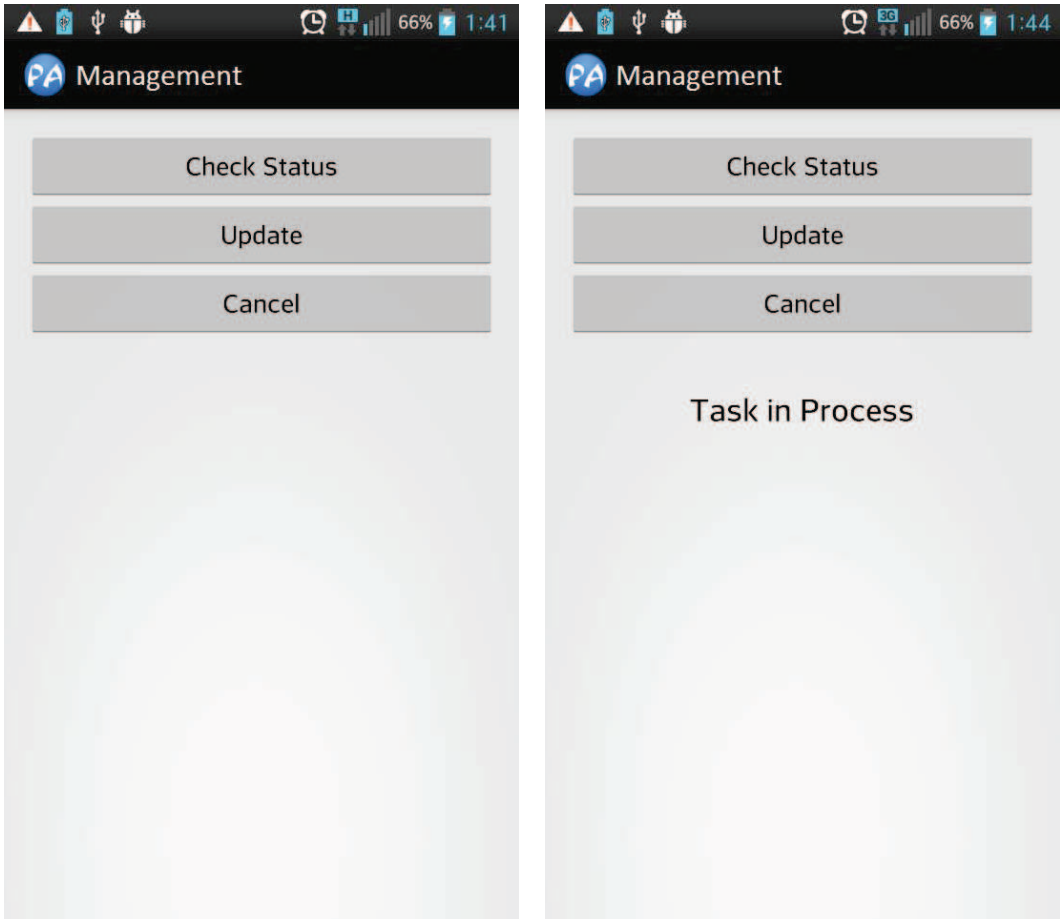
FIG. 6.2. *SAaaS4Mobile resource reservation and expiration timeout setting*

, and ***Parameter/ Statistic***	**GetCapabilities**	***DescribeTasking***	***Submit***	***Observation Access***
	ms	ms	ms	ms
μ	383.3	381.52	586.53	345.66
σ	11.2	12.1	20.5	9.7

TABLE 6.1
Basic operations response time obtained through the experiments.

All the parameters thus obtained through the evaluation are reported in cf. Table 6.1 . From these values we can argue that the most time-consuming operation is the *Submit* one, while the *Observation Access* request is that with the lowest delay. It could be also observed that both *GetCapabilities* and *DescribeTasking* have more or less similar performance. Thus, a whole workflow as the one depicted in Figure 5.1, made up of a sequence of invocations for the four aforementioned operations, has at least a response time of about 1700 ms.

These preliminary values strongly encourage us in furthering the development of the SAaaS4Mobile framework, since they serve as a foundation for assessing the feasibility of the SAaaS approach. A further and more comprehensive use case development, based on this preliminary implementation, is ongoing.

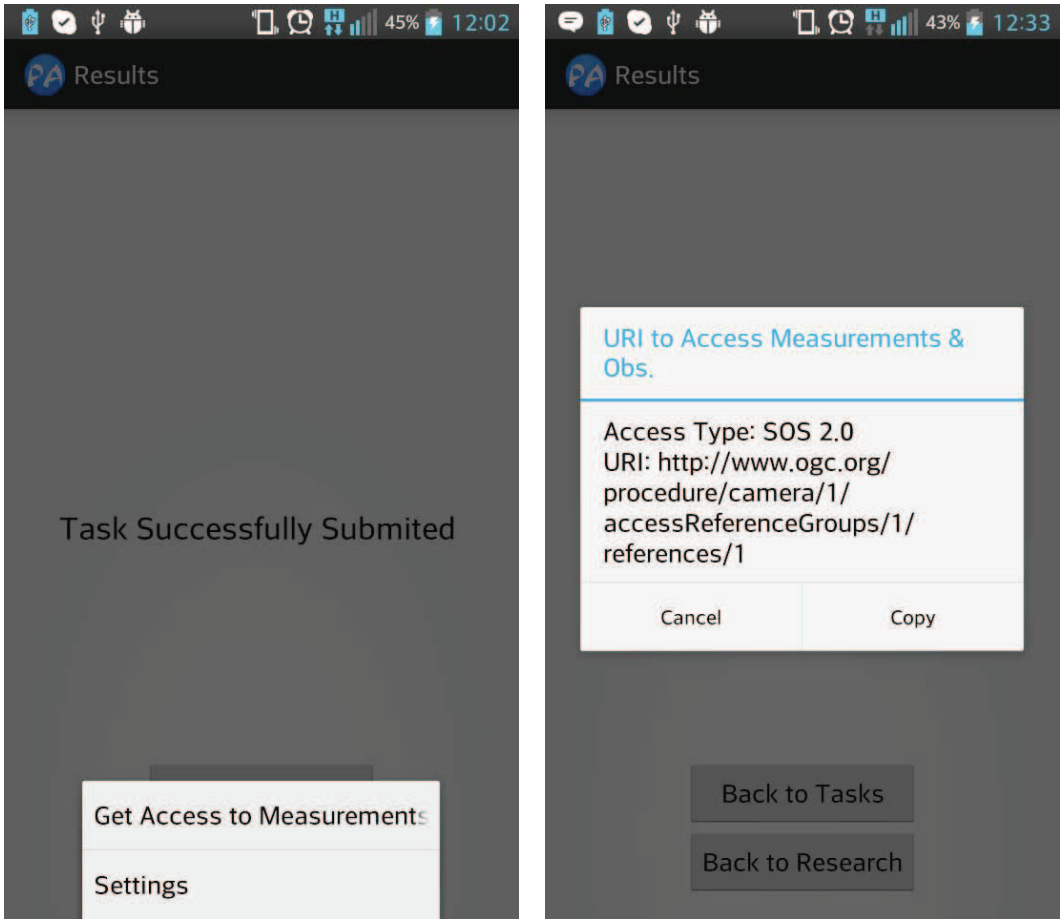


(a) Reservation (b) Expiration timeout

FIG. 6.3. *SAaaS4Mobile resource reservation and expiration timeout setting*

7. Conclusions. Aim of this paper is to continue presenting and discussing the feasibility of a sensing Cloud, able to actively involve devices, personal or standalone as well as grouped into specific administration domains such as sensor networks, either mobile or static. According to this vision, sensing and actuation resources, shared by device owners and administrators in a volunteer contribution fashion, are gathered by sensing Cloud providers to be provided on-demand, elastically, according to end-user requirements. This approach has been formalised into the Sensing and Actuation as a Service paradigm that, similarly to IaaS for compute Clouds, aims at providing actual, even if virtual, (sensing) resources. With regards to mobiles, this perspective complements and extends the one relative to mobile Clouds, where mobiles are just clients of Cloud-powered services, by actively involving them into a wide sensing infrastructure accessed and provided as a service. This way, the SAaaS paradigm lays at the intersection between the IoT and the service oriented/utility/Cloud computing fields.

To implement such a sensing Cloud several functionalities such as abstracting, virtualising, enrolling, collecting, discovering and managing (sensing and actuation) resources are required. Abstraction is a very basic one, since all nodes should be able to join the sensing Cloud and to communicate/interoperate, thus they should provide a uniform, abstract interface to underlying physical resources. In this paper we mainly focus on mobiles, dealing with issues related to sensing resource access and management through the SAaaS4Mobile



(a) Access request (b) Results URI

FIG. 6.4. *SAaaS4Mobile oservation access*

framework. In particular, the design of the SAaaS4Mobile Hypervisor module is tailored on mobiles' unique features, mainly specifying its blocks such as the Planning Agent, together with the communication layer inside the Node Manager and with the low-level access to devices the Translation Engine abstracts away.

Static and dynamic behaviour of these components have been described, both detailing their architecture and focusing on their interactions as well as on those between the end-user and the contributing node, i.e. the SAaaS4Mobile Hypervisor client. We also described these interactions and commands, from the angle of our prototype implementation on Android smartphones, also testing some on them through a proof of concepts demonstrating the feasibility of the approach.

Further endeavours are going to investigate and explore aspects related to virtualization, as well as to port SAaaS implementations to SNs. We are also eager to spend efforts over use cases and application scenarios, especially to carry out useful evaluations on performance, trying to validate the SAaaS approach by uncovering outstanding advantages and exploring unique features, by extending the current SAaaS4Mobile implementation. Moreover, we are also investigating on further developments of the approach in context of IoT, considering it as the implementation of a utility vision for the IoT paradigm, able to support novel, up and coming trends such as crowdsensing.

REFERENCES

[1] K. Aberer, M. Hauswirth, and A. Salehi, *Infrastructure for data processing in large-scale interconnected sensor networks*, in Mobile Data Management, 2007 International Conference on, may 2007, pp. 198 –205.

[2] M. Avvenuti, P. Corsini, P. Masci, and A. Vecchio, *An application adaptation layer for wireless sensor networks*, Pervasive Mob. Comput., 3 (2007), pp. 413–438.

[3] M. Behan and O. Krejcar, *Modern smart device-based concept of sensoric networks*, EURASIP Journal on Wireless Communications and Networking, 2013 (2013), p. 155.

[4] D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and M. Scarpa, *Evaluating wireless sensor node longevity through markovian techniques*, Computer Networks, 56 (2012), pp. 521–532.

[5] Z. Chen, N. Chen, L. Di, and J. Gong, *A flexible data and sensor planning service for virtual sensors based on web service*, Sensors Journal, IEEE, 11 (2011), pp. 1429–1439.

[6] J. Clarke, J. Lethbridge, R. Liu, and A. Terhorst, *Integrating mobile telephone based sensor networks into the sensor web*, in Sensors, 2009 IEEE, 2009, pp. 1010–1014.

[7] V. D. Cunsolo, S. Distefano, A. Puliafito, and M. Scarpa, *Cloud@home: Bridging the gap between volunteer and cloud computing*, in Emerging Intelligent Computing Technology and Applications, D.-S. Huang, K.-H. Jo, H.-H. Lee, H.-J. Kang, and V. Bevilacqua, eds., vol. 5754 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 423–432.

[8] S. Distefano, *Evaluating reliability of wsn with sleep/wake-up interfering nodes*, Int. J. Systems Science, 44 (2013), pp. 1793–1806.

[9] S. Distefano, G. Merlino, and A. Puliafito, *Sensing and actuation as a service: A new development for clouds*, in Proceedings of the 2012 IEEE 11th International Symposium on Network Computing and Applications, NCA '12, Washington, DC, USA, 2012, IEEE Computer Society, pp. 272–275.

[10] S. Distefano, G. Merlino, A. Puliafito, and A. Vecchio, *A hypervisor for infrastructure-enabled sensing clouds*, in IEEE International Conference on Communications, Budapest, Hungary, June 9-13, 2013 2013.

[11] S. Distefano and K. S. Trivedi, *Non-markovian state-space models in dependability evaluation*, Quality and Reliability Eng. Int., 29 (2013), pp. 225–239.

[12] M. Fazio, M. Villari, and A. Puliafito, *Sensing technologies for homeland security in cloud environments*, in Sensing Technology (ICST), 2011 Fifth International Conference on, 28 2011-dec. 1 2011, pp. 165 –170.

[13] M. Gaynor, S. L. Moulton, M. Welsh, E. LaCombe, A. Rowan, and J. Wynne, *Integrating wireless sensor networks with the grid*, IEEE Internet Computing, 8 (2004), pp. 32–39.

[14] G. Gil, A. Berlanga de Jesus, and J. Molina Lopez, *incontexto: A fusion architecture to obtain mobile context*, in Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on, 2011, pp. 1–8.

[15] Google Inc., *Android ndk - http://developer.android.com/tools/sdk/ndk/index.html*.

[16] ———, *Android Website - http://www.android.com/*, 2013.

[17] V. Huang and M. Javed, *Semantic sensor information description and processing*, in Sensor Technologies and Applications, 2008. SENSORCOMM '08. Second International Conference on, 2008, pp. 456–461.

[18] J. Jamsa, M. Luimula, J. Schulte, C. Stasch, S. Jirka, and J. Schoning, *A mobile data collection framework for the sensor web*, in Ubiquitous Positioning Indoor Navigation and Location Based Service (UPINLBS), 2010, 2010, pp. 1–8.

[19] A. P. Jayasumana, Q. Han, and T. H. Illangasekare, *Virtual sensor networks - a resource efficient approach for concurrent applications*, in Information Technology, 2007. ITNG '07. Fourth International Conference on, april 2007, pp. 111 –115.

[20] Y. Liu, D. Hill, A. Rodriguez, L. Marini, R. Kooper, J. Myers, X. Wu, and B. Minsker, *A new framework for on-demand virtualization, repurposing and fusion of heterogeneous sensors*, in Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems, CTS '09, Washington, DC, USA, 2009, IEEE Computer Society, pp. 54–63.

[21] M. Navarro, M. Antonucci, L. Sarakis, and T. Zahariadis, *Vitro architecture: Bringing virtualization to wsn world*, in Proceedings of the 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems, MASS '11, Washington, DC, USA, 2011, IEEE Computer Society, pp. 831–836.

[22] Open Geospatial Consortium, *OGC(R) Sensor Planning Service Implementation Standard*, OGC, 2.0 ed., 2011.

[23] C. Reed, M. Botts, J. Davidson, and G. Percivall, *Ogc(r) sensor web enablement: overview and high level achhitecture.*, in Autotestcon, 2007 IEEE, sept. 2007, pp. 372 –380.

[24] L. Sarakis, T. Zahariadis, H.-C. Leligou, and M. Dohler, *A framework for service provisioning in virtual sensor networks*, EURASIP Journal on Wireless Communications and Networking, 2012 (2012), p. 135.

[25] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh, *Hourglass: An infrastructure for connecting sensor networks and applications*, tech. report, 2004.

[26] L. Sun, D. Zhang, and N. Li, *Physical activity monitoring with mobile phones*, in Toward Useful Services for Elderly and People with Disabilities, B. Abdulrazak, S. Giroux, B. Bouchard, H. Pigot, and M. Mokhtari, eds., vol. 6719 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 104–111.

[27] M. Yuriyama and T. Kushida, *Sensor-cloud infrastructure - physical sensor management with virtualized sensors on cloud computing*, in Network-Based Information Systems (NBiS), 2010 13th International Conference on, sept. 2010, pp. 1 –8.

[28] Y. Zhi-An and M. Chun-Miao, *The development and application of sensor based on android*, in Information Science and Digital Content Technology (ICIDT), 2012 8th International Conference on, vol. 1, 2012, pp. 231–234.

Edited by: Maria Fazio and Nik Bessis
Received: Nov 2, 2013
Accepted: Jan 10, 2014

DOI 10.12694/scpe.v14i4.930
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. 235–247. http://www.scpe.org

TOWARDS AN AUTOMATED BPEL-BASED SAAS PROVISIONING SUPPORT FOR OPENSTACK IAAS

PAOLO BELLAVISTA † ANTONIO CORRADI † LUCA FOSCHINI † AND ALESSANDRO PERNAFINI ‡

Abstract. Software as a Service (SaaS) applications fully exploit the potential of elastic Cloud computing Infrastructure as a Service (IaaS) platforms by enabling new highly dynamic Cloud provisioning scenarios where application providers could decide to change the placement of IT service components at runtime, such as moving computational resources close to storage so to improve SaaS responsiveness. Moreover, emergent Internet of Things (IoT) scenarios enable novel computing applications involving several heterogeneous smart objects interacting with each other. These highly dynamic scenarios call for novel Cloud support infrastructures able to automate the whole SaaS provisioning cycle spanning from resource management to dynamic IT service components placement, including software deployment, components re-activation, and rebinding operations. However, notwithstanding the core importance of these functions to truly enable the deployment of complex SaaS over IaaS environments, at the current stage only partial and ad-hoc solutions are available. This paper presents a support infrastructure aimed to facilitate the composition of heterogeneous resources, such as single Virtual Machines (VMs), DB services and storage, and stand-alone services, by automating the provisioning of complex SaaS applications over the widely diffused real-world open-source OpenStack IaaS. Collected experimental results show the effectiveness of parallel execution of deployment steps introduced by our solution and demonstrate its applicability and advantages in a real SaaS production testbed.

Key words: Cloud computing; Internet of Things; Service orchestration; OpenStack; Juju; BPEL

1. Introduction. Novel Cloud computing infrastructures consisting of worldwide fully interconnected data centers offering their computational resources as IaaS on a pay-per-use basis are opening brand new challenges and opportunities to develop novel SaaS-based applications. Moreover, during the last decade, we experienced the emergency of IoT application scenarios, where heterogeneous and ubiquitous devices, spanning from fully-fledged smartphones to wired and wireless sensors, can interact with each other and cooperate to achieve common goals of enabling new smart scenarios. The unique requirements of IoT environments (such as fast deployability, high scalability, and large-scale provisioning), together with their highly dynamic nature, call for the development of a large number of new SaaS applications exploiting the elasticity offered by novel Cloud systems. These systems are typically characterized by both agile and continuous developments and deployments as well as ever-changing service loads, and call for highly novel automatic solutions able to dynamically and continuously supervise and facilitate the whole application management lifecycle.

In recent years, the advent of new Platform as a Service (PaaS) environments, such as CloudBees, CloudFoundry, and OpenShift has simplified the provisioning of new SaaS applications over physical and IaaS-based Cloud systems [1, 2, 3]; at the same time, PaaS technologies tend to impose to the final developer fixed and well-defined software stacks (including languages and usable services), often difficult to modify and to tailor to the specific service needs. In addition, from a more technological perspective, while SaaS and IaaS solutions have been widely used and employed in the last decade even before the advent of the Cloud wave, PaaS represents a younger technology that still deserves much work to improve flexibility and interoperability between different PaaS environments, as well as in enhancing integration opportunities with other existing IaaS and SaaS ones. Focusing only on SaaS-over-IaaS solutions, enabling the management and especially the provisioning of complex SaaS applications over highly dynamic and large-scale Cloud-based IoT environments is still a difficult task that requires to solve several open management issues spanning from virtualization issues, such as Virtual Machine (VM), storage, and network virtualization, to large-scale Cloud monitoring, from optimal resource placement computation to standardization and interoperability of the different deployment frameworks and Application Programming Interfaces (APIs) adopted by various Cloud providers, and so forth.

Among all these challenging issues, the purpose of this paper is to present an architecture that offers a support for the orchestration of all the steps needed to publish a SaaS application within a Cloud IaaS. A *SaaS application* inside a Cloud environment can be viewed as a collection of opportunely configured *service components* deployed into a set of dynamically created IaaS resources. In modern datacenters, there is a high

†Dipartimento di Informatica Scienza e Ingegneria (DISI), Bologna, Italy ({paolo.bellavista,antonio.corradi,luca.foschini}@unibo.it).
‡Centro Interdipartimentale di Ricerca Industriale ICT (CIRI ICT), Bologna, Italy (alessandro.pernafini@unibo.it).

availability of computational, storage, and network resources, but it is still missing a mechanism to automatically orchestrate all the involved entities to allocate resources, to deploy and configure various software components, and to manage their interactions in order to provide the requested application. Indeed, before application providers can provide an application, they need to manually perform a set of operations (i.e., request new VMs, install and configure software) that, especially for large-scale deployments, like the ones we could obtain in IoT scenarios, could be really time consuming thus reducing the advantages of having flexible compute infrastructures.

This specific problem has already been partially addressed by some contributions in the literature; however, most of the existing efforts focus on single aspects. For instance, some proposal addressed deployment and lifecycle management of service components [4, 5, 6], while the integration of software lifecycle management as a core function of IaaS environment management supports, instead, apart a few specific seminal studies [7, 8, 10], is still widely unexplored. In this context, we claim the necessity of new fully-integrated automated SaaS provisioning facilities that start from the management of virtual resources, pass through the installation, configuration and management of software components, and end with the coordination of these components. That would be highly beneficial both for SaaS application providers, especially in highly dynamic IoT environments, to ease the realization of new SaaS applications through the composition of existing single service components in a mash-up like fashion, and for IaaS Cloud providers, by taking over all the error-prone and time-consuming deployment and configuration operations at the IaaS level.

To address all these open issues, this paper proposes a novel automated SaaS-over-IaaS provisioning support that adopts three main original guidelines. First, it provides to both IaaS Cloud providers and to SaaS application providers a tool that transparently takes over the execution of software deployments and updates with almost no need for human intervention. Second, it proposes a general automated application provision support that integrates with state-of-the-art technologies, such as the highly interoperable OpenStack IaaS and the standard Business Process Execution Language (BPEL), to ease the definition of all main deployment, configuration, and deployment monitoring steps. Third, our prototype has been implemented as an open-source tool based on the open-source OpenStack Cloud platform and is made available to the Cloud community. Finally, in order to better underline the benefits and original aspects of the proposed solution and to demonstrate the effectiveness of our solution, the paper presents an experimental evaluation based on a realistic SaaS application provisioning scenario on top of an open-source testbed based on OpenStack.

The remainder of this paper is organized as follows. In Sect. 2, we give an overview of related work in the literature. In Sect. 3, we introduce needed background material about all main involved standards, technologies, and support tools; in Sect. 4, we present our framework and outline its main components; in Sect. 5, we provide some implementation details about our presented architecture. Finally, in Sect. 6 we show collected experimental results. Conclusions and directions of future work end the paper.

2. Related works. The on-demand provisioning of services and resources in distributed architectures has been deeply investigated in recent years. For the sake of space limitations, we will focus on two research directions only: we start with works that provide solutions for the deployment and lifecycle management of software components; then we move towards solutions that, closer to our proposal, enable automated provisioning of applications by integrating software lifecycle as part of the wider Cloud IaaS management operations.

Focusing on the first research direction, the design, deployment, and management of software components can be challenging in systems distributed on a large scale, and several different systems provide solutions to automate these processes. The work depicted in [4] presents a system management framework that, given a model of configuration and lifecycle, automatically builds a distributed system. Similarly, authors of [5] introduce a model-based solution to automatically configure system specifications and provide this system on-demand to the user. Finally, in [6], authors presented a solution to face change management issues; this solution aims to automate all the steps required to handle software or hardware changes to existing IT infrastructures, with the goal of an high degree of parallelism. All these solutions provide the automation of the deployment and management of software components, so relieving administrator of the burden of manually configure distributed systems; however, they only focus on the deployment of software components and do not consider virtual infrastructure management, that instead assumes a central role in Cloud environments.

Along the second research directions, some seminal works have started to analyze the automated provisioning

of applications in Cloud systems. The solution presented in [7] describes a multi-layer architecture that enables the automated provisioning and management of cloud services; with this solution users can select a service from a catalog of service templates, then the service can be configured by the user and deployed automatically. Authors of [8] present a solution for on-demand resource provisioning based on BPEL [9]. This solution extends BPEL implementations with the possibility to schedule workflow steps to VMs having a low load and the possibility to add new VMs on-demand in peak-load situations. Both solutions focus on one of the most challenging aspects of Cloud computing, i.e., the capability to request and use computational resources in a small lapse of time, resulting in a fast performance increment and in a decrease of management costs. The works depicted in [10] and [11] propose similar architectures for a generic provisioning infrastructure based on BPEL. These solutions allow SaaS application providers to define generic provisioning workflows independent from the underlying provisioning engines by enabling the possibility to automate the component-to-workflow matching process; they also supports dynamic provisioning flows in order to face peak-load situations by allocating additional resources at runtime. At the same time, these approaches focus more on the theoretical part of the management process and leave out of the scope of the work possible implementation issues and analysis of additional overhead introduced by the proposed solutions. Finally, another very interesting effort, also because complementary to ours, is the one presented in [12] that aims to standardize both topology and orchestration specifications for Cloud applications with a goal to make SaaS applications and their management portable across different IaaS Cloud providers.

3. Background. This section introduces some background knowledge to provide a better understanding of the area. Section 3.1 presents Cloud IaaS environments and provides needed details about the standard-de-facto OpenStack IaaS [13]. Section 3.2 presents Juju, a scripting-based tool to ease the deployment of service components [14]. Finally, Section 3.3 gives some needed background material about the BPEL standard that we use to orchestrate the whole application provisioning process through the definition of proper workflows [9].

Before starting, let us introduce some terminology about the three main types of actors in Cloud systems: *Application users*, *Application providers*, and *Cloud providers*. Application users are the final clients that require access to particular online SaaS application and use its resources. Application providers build and expose SaaS applications, typically composed by several service components, to the end users, and tend to externalize the execution of their own services to avoid the deployment of costly private IT infrastructure. Finally, Cloud providers supply application providers with resources on a pay-per-use fashion, in order to let them execute their applications over their IaaS-based environment. In this paper, we will focus mainly on the application providers and on how they interact with Cloud providers to enable, declare, and monitor the provisioning of complex applications consisting of multiple service components.

3.1. OpenStack. OpenStack is an open-source project for building and managing private and public Cloud infrastructures [13], proposed and promoted by NASA and Rackspace in 2010. OpenStack belongs to the category of Infrastructure as a Service (IaaS) systems, whose goal is to provide resources, such as virtual machines, virtual storage blocks, etc., on-demand from large pools installed in datacenters. OpenStack is based on a very flexible architecture supporting a very large set of hardware devices and hypervisors (i.e. Hyper-V, KVM, ESX, etc.) and even small businesses are allowed to deploy their own private Cloud because of the open-source nature of this solution. However, OpenStack still lacks a monitoring and dynamic reconfiguration mechanism to favor a dynamic deployment of applications on a large scale, thus requiring a manual management to tailor specific scenarios and deployments.

OpenStack manages computation, storage and networking resources on the Cloud in order to provide dynamic allocation of VMs [13]. OpenStack is based on five main services: the first one, called Nova, to manage both computational and networking resources; the second one, named Glance, to manage and provide VMs images; the third one, Neutron to manage network resources, and, finally, Swift and Cinder to manage storage resources. To better understand our work, we provide a more detailed description of Nova service.

Nova manages the creation and the configuration of VMs, starting from images stored in *Glance* catalog. Nova does not implement any virtualization software, rather it defines some standard interfaces to control the underlying virtualization mechanisms. All the requests made to Nova components are sent through RESTful APIs to *nova-api* that acts as a front-end to export all OpenStack IaaS functionalities, such as VM creation and termination, through Web Services. To maintain compatibility towards multiple vendors and to facilitate the

migration toward different Cloud providers, OpenStack also supports Amazon EC2 APIs to deploy applications written for Amazon Web Services with a minimal porting effort [15]. In the following, we report several other details about the main Nova services.

Nova-compute service, running on every node in the Cloud, launches and configures VMs within a certain physical host. It communicates with the underlying hypervisor to instantiate and terminate VMs and to obtain load statistics as well as performance metrics of VMs. OpenStack supports a wide range of hypervisors, but the most commonly used hypervisor is KVM, due to its good performance and its full support toward the virtualization of x86 architectures.

Nova-network service manages all the aspects related to network management. This service makes it possible to create virtual networks that allow communications between different instances of VMs. A private IP is assigned to every VM during boot, but it is also possible to assign it a public IP in order to make it accessible over the Internet. All networking functionalities are moving towards the OpenStack service Neutron. This service offers the possibility to create networks that can be associated to different tenants; it is also possible to create virtual routers to enable communication between two or more VMs belonging to different tenant networks. Thus, networks can be seen as resources available in the Cloud and Neutron can be considered as a Network as a Service (NaaS).

Finally, *nova-scheduler* service determines on which node a VM should be booted. Actually this service offers only a small set of simple scheduling policies, such as selecting the least loaded host or randomly selecting a host. Even if OpenStack offers a scheduler mechanism to choose where a VM should be booted, it does not provide any dynamic mechanism to migrate running VMs based on the current host load.

To show the interactions between OpenStack services, we introduce a simple VM instantiation use case (see Fig. 3.1). The current state of the entire Cloud is maintained in a SQL server; periodically each nova-compute service running on a certain node updates the SQL server with load information about that node (step 0). When a user requests the instantiation of a new VM through the RESTful APIs (step 1), the nova-api service sends a request to the nova-scheduler service (step 2) to determine on which host the new VM should be launched. In this step, the scheduler queries the database in order to obtain a list of available hosts along with their load information (step 3), and then selects one of them (step 4) according to the chosen policy. Finally, the scheduler sends a VM instantiation request to the nova-compute service running on the selected node (step 5) that requests network configuration parameters to the network service (step 6).

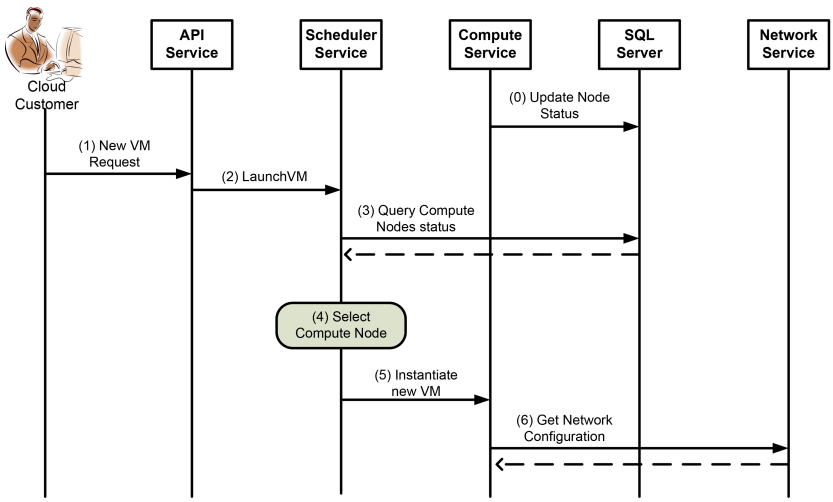


FIG. 3.1. *VM instantiation in OpenStack*

3.2. Juju. Juju is a tool for the deployment and the orchestration of services that grants the same ease of use we can see in some widely used packet management systems such as Advanced Packaging Tool (APT) or Red Hat Package Manager (RPM) [14].

Juju focuses on the management and deployment of various service units and components needed to provide a single application, by taking over the configuration and installation of required software on the VMs where these service components will be deployed. Juju allows independent service components to communicate through a simple configuration protocol. End-users can deploy these service components inside the Cloud, in a similar way they can install a set of packets with a single command. As a result, it is possible to obtain an environment consisting of multiple machines whose service components cooperate to provide the requested application.

Juju is independent from the underlying Cloud Infrastructure Layer and supports several Cloud providers such as OpenStack, Amazon Web Services, HP Cloud, Rackspace, etc. Thus, it is possible to migrate a service component between different Clouds with minimal re-deploy effort.

A service component represents an application or a group of applications integrated as a single component inside a Juju environment that can be used by other components in order to build an higher level application. In this paper we consider the use case where we provide WordPress, an open-source platform to create, manage, and create dynamic Web site [16], by configuring and orchestrating two distinct service components: a service component exposing the MySQL database needed by WordPress, and another service component running the WordPress engine. A service component instance is called Service Unit and it is possible to add more of these Service Units to the environment in order to scale the whole system, thus reducing the load on each VM.

Three main concepts are at the basis of services publication: *charms*, *hooks* and *relations*.

A *charm* encapsulates the logic required to publish and manage a service component inside a Juju environment. A charm provides the definition of a service component, including its metadata, its dependences on other service components, the software packets we need to install in a VM, along with the logic needed to manage the service component. Through the definition of a charm, it is possible to define the functionalities exposed by the service component and, if we are dealing with a composed service, all the sub-services required.

Hooks are executable files used by Juju to notify a service component about changes related to its lifecycle or about other events happened inside the environment. When a hook is executed, it can modify the underlying VM (i.e. it could install new software packets) or it can change relations between two or more service components.

Finally, *relations* allow the communication between different service components. Relations are defined inside a charm to declare the interfaces needed/exposed by a service component, that are offered/used by another service component. Low level communications between service components are based on TCP sockets.

The *environment* is a fundamental concept at the basis of Juju: it can be seen as a container where service components can be published; environments are managed through a configuration file where it is possible to define some configuration parameters such as used Cloud provider, IP address of the Cloud provider, authentication credentials, etc.

It is possible to execute an environment through the bootstrap operation exposed by Jujus API. The bootstrap operation initialize the system, instantiating a VM that will act as the controller node of the environment. Zookeeper and Provisioning Agent are two of the main software components executed on controller node. Zookeeper can be viewed as a file systems that stores all the information about the environment, while Provisioning Agent interacts with the underlying Cloud provider in order to instantiate and terminate VMs where service components are going to be deployed.

3.3. BPEL. BPEL is the de facto standard to define business processes and business interaction protocols [9]. The BPEL language, based on XML, allows to express the orchestration of multiple Web Services by defining business interactions modeled after a sequence of message exchanges between involved entities. A BPEL document contains the control logic required to coordinate all the Web Services involved in a workflow.

BPEL provides many language constructs and mechanisms to define a sequence of activities like *invoke*, *receive* and *reply*, parallel and sequential execution, transactional execution of a group of activities, and exception handling. A *partnerLink* is an important construct defined by BPEL to represent an external service that is invoked by a process or that invokes the process itself.

A BPEL engine elaborates a BPEL document, by defining an orchestration logic, and consequently executes all the activities according to the order defined by the logic. Typically, a BPEL engine exposes the business process through a Web Service interface that can be either accessed by Web Service clients or used in other business process. One of the main advantages of BPEL is that the several activities of a business process can be executed simultaneously, instead of imposing a sequential execution.

4. Architecture. This section presents our architecture proposal to face all the main service orchestration challenges described in the previous sections: the proposed architecture provides the support to orchestrate all the steps involved in the publication of an application inside a Cloud platform, starting from the instantiation of required VMs to the deployment of required software components, together with the definition of their relationships. First, we briefly introduce this architecture and then we give a more in deep description of its components.

The proposed architecture is easily extensible, due to its multi-layer nature; it allows to arbitrarily manage the software components that form an application, and to use several Cloud providers. Starting from requests asking for application provisioning sent from application providers, it is possible to automatically satisfy their requests by monitoring all the steps involved in the application publication and notifying application providers about the progress of their request.

The proposed architecture (see Fig. 4.1) consists of a *Cloud Infrastructure Layer* and a *Service Orchestrator Layer* that, in its turn, we logically divided in two sub-layers: an *Abstraction Layer* and an *Orchestration Layer*.

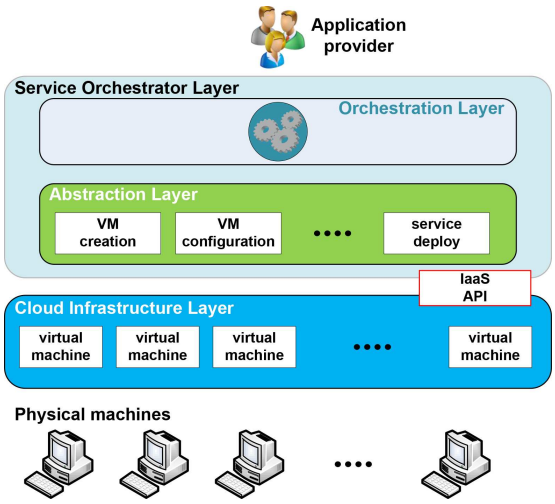


FIG. 4.1. *Proposed architecture*

The Cloud Infrastructure Layer represents the virtual resources provided by the Cloud infrastructure through the IaaS API: it contains VMs instances and defines the APIs required to create, configure and destroy VMs used by upper layers; it also offers a connection mechanism in order to grant access to VMs. In our implementation, we choose to use OpenStack as Cloud Infrastructure Layer, as it is a widely adopted open-source solution; at the same time, thanks to the highly flexible nature of our architecture, it is possible to use any other Cloud provider.

The Orchestration Layer and the Abstraction Layer compose together the Service Orchestrator Layer. It is the composition of these two layers that makes it possible to create an orchestration support. Once the application provider has sent a request, this layer will coordinate and execute all the activities to satisfy that request, by opportunely configuring and communicating with the VMs provided by the Cloud Infrastructure Layer.

Abstraction Layers goal is hiding the complexity of the underlying Cloud Infrastructure Layer by providing a high level interface to the Orchestration Layer which encapsulates the functionalities offered by the Cloud Infrastructure Layer. This abstraction mechanism obtains a highly flexible architecture working with several Cloud providers. The functionalities exposed by this layer are useful to manage the entire VM lifecycle, in addition to the services offered by that VM. This makes it possible to create a VM with a chosen operating system and install on it all the software components required to build a service. Moreover, it is also possible to add relationships between different services in order to allow them to cooperate. Let us introduce an example to better understand the functionalities. If we want to build a service exposing a dynamic web site, we need to instantiate and deploy two sub-services: a web server and a database to store all objects and data required

by the web server. To deploy this scenario, the Abstraction Layer will create two VMs (one for the web server and the other one for the database), install all the required software packages, and configure and start the two services. However, in order to publish a working web server, these services need to communicate to each other. This can be done by defining a relationship between the two services and specifying the functionalities exposed by each service along with the required functionalities. It is essential that the Abstraction Layer could access the VMs where the two services are deployed in order to monitor and, possibly, reconfigure the services; this is achieved by establishing SSH tunnels to VMs.

The Orchestration Layer represents the orchestration engine inside our architecture. When an application provider submits a request to this layer, it coordinates and orchestrates all the steps required to automatically provide the application provider with the requested application. Every request received by the Orchestration Layer contains a description of the required application, that can be seen as a model defining the service components that compose the application, along with the description of their relationships to determine how they must mutually interact. Typically, many activities are involved in exposing an application, so this layer needs to manage transitions between these activities, by taking into account the dependencies between service components as shown in Fig. 4.2. These dependencies represent the synchronization points between operation sequences executed inside a workflow.

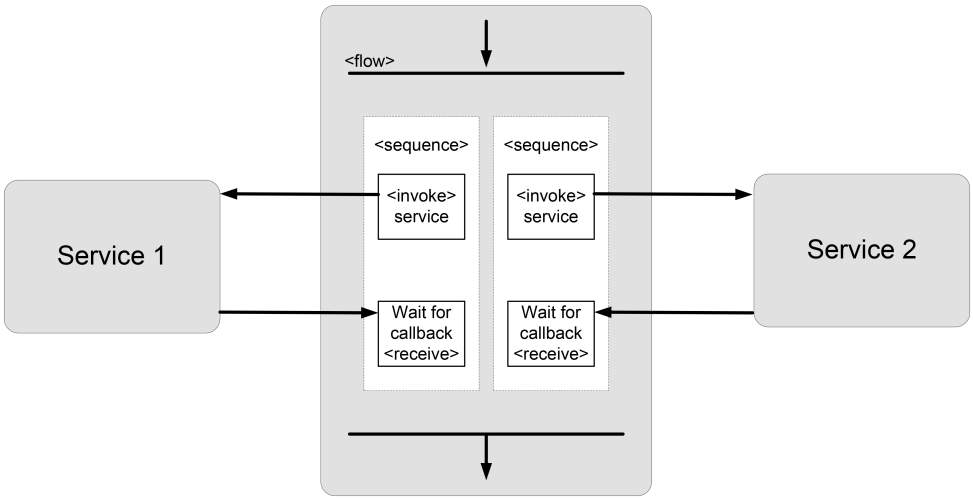


FIG. 4.2. *Typical Orchestration Layer workflow*

Going back to our previous example, it is impossible to publish a web server before the database is ready, because it would lack the required support to manage data. When the database is ready and the web server has been deployed, we can specify the relationship between these two software components. The Service Orchestrator Layer deploys those service components in parallel, monitoring the involved steps; that allows to simultaneously deploy several service components. In our solution, we implement this layer by using a BPEL engine.

5. Implementation Details. This section provides some implementation insights about our solution, based on both proprietary and ad-hoc software. Our presentation will follow a bottom-up approach, starting from the physical layer up to the Orchestration Layer. For the Cloud Infrastructure Layer, we have chosen OpenStack due to its highly flexible and open-source nature; in particular, we used the latest Havana release. Atop OpenStack, we use Juju to implement our Abstraction Layer: functionalities exposed by Juju encapsulate APIs provided by OpenStack, so we opportunely configured Juju environment in order to work with OpenStack, hiding these configuration details to the application provider. Other open-source service management tools, such as Puppet [17] or Chef [18], could be used to implement the Abstraction Layer; we chose to use Juju because it is a very recent solution, continuously evolving with the introduction of new useful features. The Orchestration Layer, using Juju charms, enables the composition of complex applications and offers monitoring facilities through the monitoring events forwarded by Zookeeper. The Orchestration layer represents the engine

of our support towards services orchestration: this layer makes it possible to coordinate the publication of SaaS applications, defining reusable and modular workflows.

Fig. 5.1 shows how the architecture layers interact with each others in order to provide a generic application composed by two different service components. Starting from a BPEL workflow defined by a Cloud provider, the BPEL engine will send two simultaneous requests to Juju so as to deploy these service components (step 1 in Fig. 5.1). Juju will then ask OpenStack to create two VMs, and, after the VMs have been booted, it will download and install software packets on them (steps 2 and 3). Once the two service components have been configured, the BPEL engine will ask Juju to add a relation between them (step 4); finally, Juju will opportunely configure these components in order to let them cooperate.

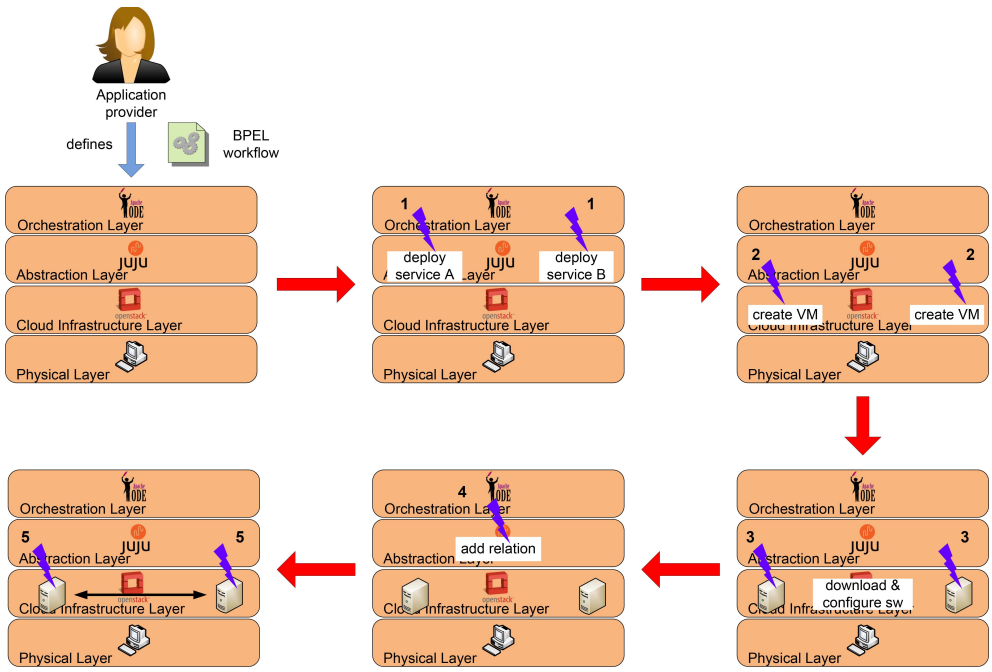


Fig. 5.1. *Interactions among architecture layers*

In particular, we used our Cloud support to implement the case study of a WordPress platform composed by two service components: a MySQL database and a WordPress engine running on a web server, each one deployed on a separate VM. Let us stress that simple services, such as this one and the Wiki service considered in the experimental results, are becoming more and more relevant in IoT scenarios to ease the publication of collected smart data by using Web-enabled and widely accessible data portals and front-ends.

In order to deploy a working WordPress platform, first we need to deploy the database service component and the WordPress engine, and then to add a relation between them to let them cooperate. We mapped all these steps into the BPEL workflow shown in Fig. 5.2.

The BPEL process, defined as an XML document, contains all the references to the external Web Services employed in the workflow; this can be done by populating the `<partnerLinks>` section. In our case study, we inserted references to DeployWS and AddRelationWS, to let the BPEL engine invoke them. These two Web Services represents respectively the Web Service used to deploy a service component, and the Web Service used to add a relation between two already deployed service components. The BPEL engine will also fill the request sent to DeployWS with the name of the service component that need to be deployed. BPEL constructs allow to execute the deployment of MySQL and WordPress service components (namely, two different instances of the DeployWS, see Fig. 5.2) in parallel on different VMs, and, through the definition of synchronization points, it is possible to orchestrate them. In particular, we use BPEL `<flow>` construct to achieve parallelism. A `<flow>` terminates its execution only when all activities included inside this tag have completed: in our case study, the

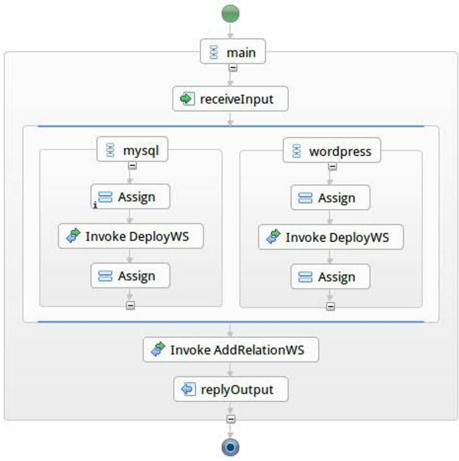


FIG. 5.2. *BPEL workflow*

completion of `<flow>` activity will occur only after both WordPress and MySQL have been deployed. Only at this time, we can invoke AddRelationWS to add a relation between these two service components.

We encapsulated the functionalities exposed by Juju, to deploy and monitor a service component inside the Web Services published on Apache Axis2. The name of the service component that needs to be published is specified inside the request sent to the Web Service. DeployWS is realized by two Java classes: Executor, that invokes `juju deploy` command in order to deploy the service component, and DataMonitor, that manages ZooKeeper events in order to monitor the progress of the request. The following figure shows an excerpt of the WSDL file relative to DeployWS (see Fig. 5.3). AddRelationWS invokes `juju add-relation` command and communicates the result of this operation to the BPEL Engine.

```
<wsdl:message name="deployWSRequest">
  <wsdl:part name="parameters" element="ns:deployWS"/>
</wsdl:message>
<wsdl:message name="deployWSResponse">
  <wsdl:part name="parameters" element="ns:deployWSResponse"/>
</wsdl:message>
<wsdl:portType name="DeployWSPortType">
  <wsdl:operation name="deployWS">
    <wsdl:input message="ns:deployWSRequest" wsaw:Action="urn:deployWS"/>
    <wsdl:output message="ns:deployWSResponse" wsaw:Action="urn:deployWSResponse
        "/>
  </wsdl:operation>
</wsdl:portType>
```

FIG. 5.3. *DeployWS WSDL code*

In order to publish WordPress and MySQL services, it is necessary to write the corresponding charm to be memorized inside the bootstrap node and sent, during the creation of a VM, to the node where that service component will be deployed. When deploying a MySQL service component, the hook install will be executed to download and configure MySQL related packets, and finally to start the service component. In the same way, all these steps will be repeated when deploying a WordPress service. After deploying MySQL and WordPress service components, the BPEL workflow adds a relation between these service components, by executing the respective relation-joined hooks. The relation-joined script relative to WordPress will write, in the WordPress

configuration file, a reference to the host where MySQL database is running, together with the credentials to access the database. Fig. 5.4 shows an excerpt of the WordPress relation-joined hook used in our tests.

```
database=`relation-get database`
user=`relation-get user`
password=`relation-get password`
host=`relation-get private-address`
juju-log "Writing wordpress config file $config_file_path"
# Write the wordpress config
cat > $config_info_path << EOF
<?php
define('DB_NAME', '$database');
define('DB_USER', '$user');
define('DB_PASSWORD', '$password');
define('DB_HOST', '$host');
define('SECRET_KEY', '$secret_key');
define('WP_CACHE', true);
```

FIG. 5.4. *Juju hook script*

6. Experimental Results. We tested our solution on a Cloud testbed environment at our campus, by considering two different use cases. The first one is more simple and realizes the implementation use case detailed in the previous section, while the second one is more complex and represents a more realistic IoT SaaS application with higher performance requirements.

Starting with the first WordPress use case, the physical Cloud testbed consists of 3 physical Linux boxes with Intel Core 2 Duo E7600 at 3.06 GHz and 4 GB RAM, connected through two 1 Gbps LANs, and running Linux Ubuntu 13.04. Fig. 6.1 shows the Cloud infrastructure and the software components deployed on it: this virtual infrastructure consists of 3 VMs running Linux Ubuntu 12.04; Juju bootstrapping node has been deployed on the first VM, while the remaining VMs, were used, respectively, to deploy a MySQL database and a web server running a WordPress engine.

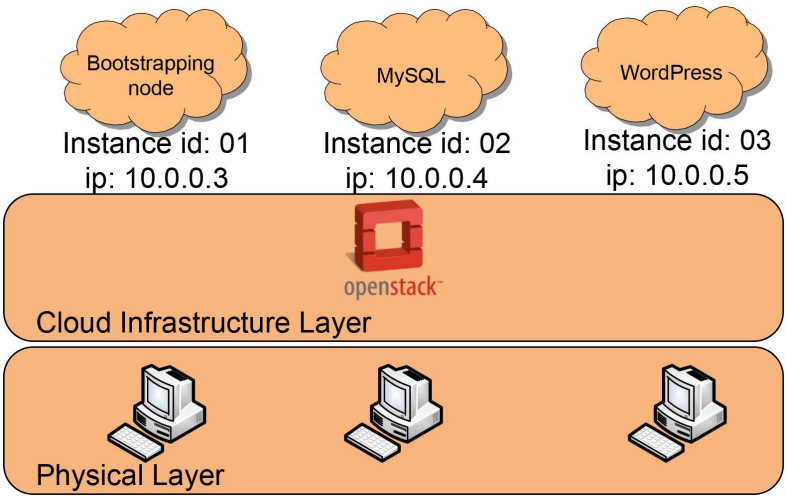


FIG. 6.1. *Testbed deployment - first scenario*

To demonstrate the efficiency of our solution, we ran two series of tests to measure the time needed to deploy a working WordPress platform. In the first series, we sequentially executed Juju commands in order to

deploy MySQL and WordPress, and, then, we added a relation between them. Instead, in the second series, we used the Orchestration Layer, in order to achieve a parallel deployment of MySQL and WordPress. All the measures were taken in a stable system, after the deployment of Juju bootstrapping node; we have repeated 30 runs for each test and we report both estimated average and standard deviation.

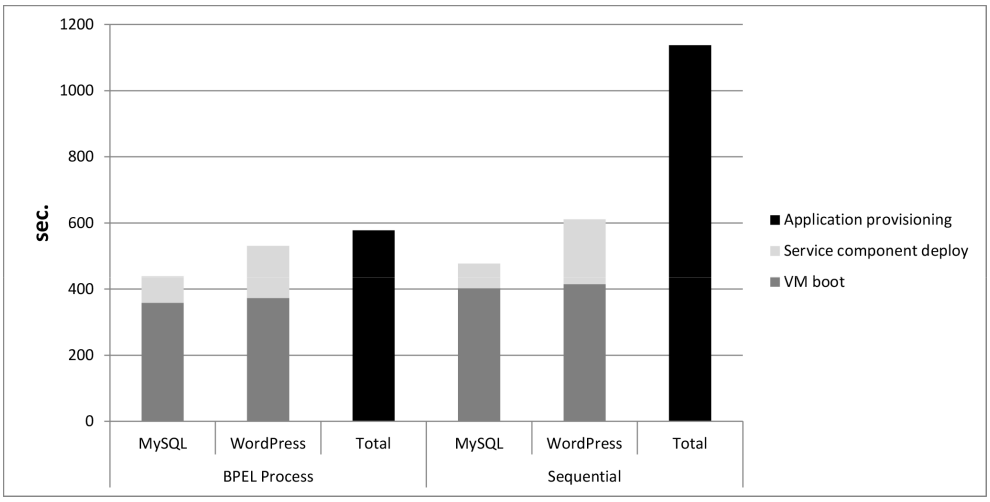


Fig. 6.2. *Deployment time - first scenario*

As we can see in Fig. 6.2, thanks to the parallel deployment of MySQL and WordPress, the overall time needed to deploy a working WordPress platform halves the time measured when deploying it sequentially. The average runtime of our tested BPEL process, including the time needed to instantiate a new VM, was about 578.4 seconds, with a standard deviation of 55.3 seconds. Instead, when using sequential Juju commands to deploy the service components, we measured an average runtime of 1138.6 seconds with a standard deviation of 115.8 seconds.

To challenge our support with a more realistic use case, we repeated all the tests described above with a more complex deployment of a multi-tier SaaS application consisting of four different service components: a service component providing a MySQL database; a service component running MediaWiki [19], an open source platform used to create wiki websites such as Wikipedia; a service component running Memcached [20], used to provide MediaWiki with a caching service, and finally a service component running HAProxy [21], an high performance load balancer for TCP/HTTP-based application. Fig. 6.3 shows the Cloud infrastructure and the software components deployed on it: this virtual infrastructure consists of 5 VMs running Linux Ubuntu 12.04; Juju bootstrapping node has been deployed on the first VM, while the remaining VMs, were used, respectively, to deploy a MySQL database, a web server running MediaWiki, a Memcached distributed memory object caching system, and a HAProxy loadbalancer. After deploying these services, three relations are added, respectively between MediaWiki and MySQL, between MediaWiki and Memcached, and between MediaWiki an HAProxy.

As shown in Fig. 6.4, the average runtime of our tested BPEL process, including the time needed to instantiate a new VM, was about 832.8 seconds (standard deviation 56.22 seconds) that significantly lowers the average runtime of 1780.8 seconds (standard deviation 119.88 seconds) needed for the sequential deployment.

So, we can conclude that the parallel execution of many processes can balance the overhead introduced by the invocation of Web Services, by the BPEL engine execution, and by execution of multiple deployment operations (potentially concurrent over the same physical host), with the enhanced performance due to parallelism; these advantages are more and more sensible as the complexity of the SaaS application to deploy increases.

7. Conclusion and Future Works. In this paper, we presented and experimentally validated a management support to automate the provisioning of complex SaaS applications over Cloud based infrastructures. Due to BPEL-based orchestration, our solution can achieve high expressivity in the definition of the application provisioning logic, including not only deployment issues, but also advanced monitoring of service component

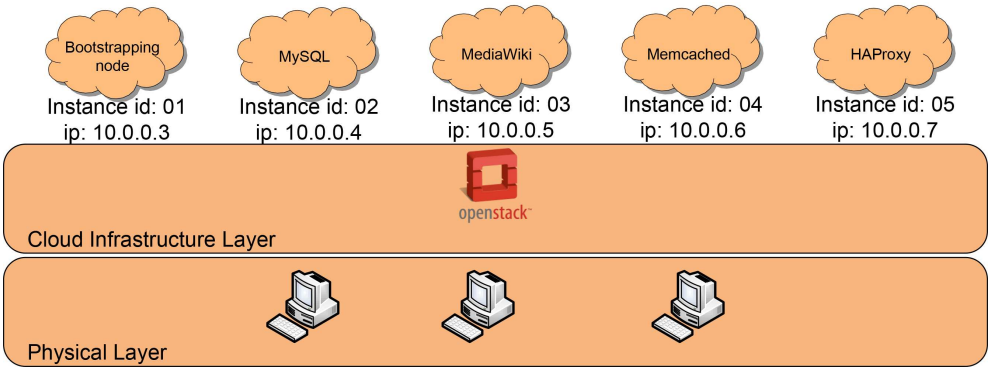


FIG. 6.3. *Testbed deployment - second scenario*

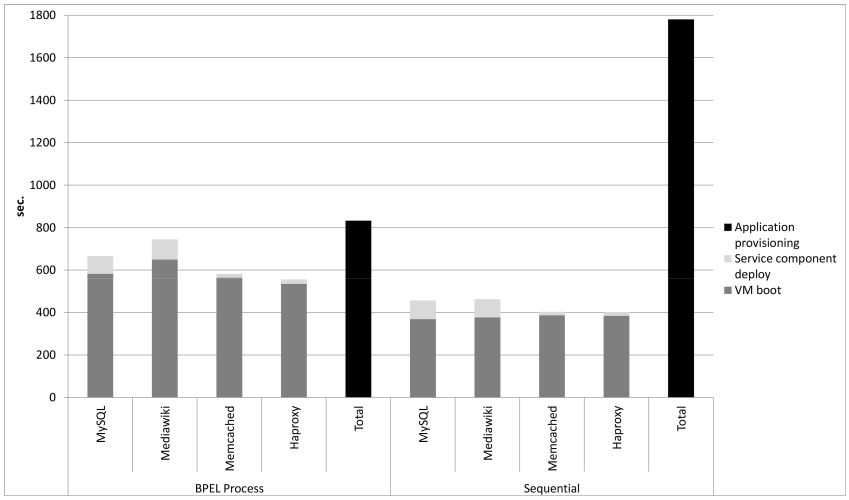


FIG. 6.4. *Deployment time - second scenario*

status. Moreover, it enables concurrent execution of parallelizable service component deployment steps, thus significantly reducing the time needed to activate complex SaaS applications in large-scale Cloud environments, particularly in IoT scenarios, where the highly dynamic nature of these environments often requires fast applications provisioning. Experimental results showed the effectiveness of the realized support that introduces a limited overhead by granting a drastic reduction of the provisioning time when deployment steps can be executed in parallel. Moreover, the use of BPEL and workflow processes enables a higher degree of flexibility and reusability of our framework; indeed, already existing provisioning workflows can be reused to provide new SaaS applications. Encouraged by these results, we are considering several future directions: on the one hand, we are currently integrating our new application provisioning facilities with our IaaS runtime monitoring and management support [22]; on the other hand, we are developing an automatic application live-migration support to move the whole application, including all needed service components and relations, from local private Cloud IaaS to public ones, by dynamically re-binding all needed virtual resources therein; finally, we are implementing a mechanism to define multi-tenant network infrastructures and to provide isolation for multi-tenant SaaS applications deployed atop them.

Acknowledgments. This research was partly funded by CIRI, technology transfer center for ICT, of the University of Bologna; we also thank CINECA for its support.

REFERENCES

[1] *CloudBees* home page, http://www.cloudbees.com/. Last accessed: June 2013.
[2] *CloudFoundry* home page, http://www.cloudfoundry.com/. Last accessed: June 2013.
[3] *OpenShift* home page, https://openshift.redhat.com/. Last accessed: June 2013.
[4] P. Goldsack et al., *The SmartFrog configuration management framework*, ACM SIGOPS Operating Systems Review, 43(2009), pp. 16–25.
[5] S. Singhal, M. Arlitt, D. Beyer, S. Graupner, V. Machiraju, J. Pruyne, J. Rolia, et al., *Quartermaster A Resource Utility System*, in Proceedings of 9th IFIP/IEEE International Symposium on Integrated Network Management, 2005, pp. 265–278.
[6] A. Keller, J. L. Hellerstein, J. L. Wolf, K. Wu and V. Krishnan, *The CHAMPS system: change management with planning and scheduling*, in Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), 2004, pp. 395–408.
[7] J. Kirschnick, J. M. Alcaraz Calero and N. Edwards, *Toward an architecture for the automated provisioning of cloud services*, IEEE Communications Magazine 48 (2010), pp. 124–131.
[8] T. Dornemann, E. Juhnke and B. Freisleben, *On-Demand Resource Provisioning for BPEL Workflows Using Amazon's Elastic Compute Cloud*, in Proceedings of 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09), 2009, pp. 140–147.
[9] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith,S. Thatte, I. Trickovic and S. Weerawarana, *Business Process Execution Language for Web Services Version 1.1.* 1.1 Edition. Microsoft, IBM, Siebel, BEA, and SAP (2003).
[10] R. Mietzner and F. Leymann, *Towards provisioning the cloud: On the usage of multi-granularity flows and services to realize a unified provisioning infrastructure for saas applications*, in: IEEE Congress on Services - Part I, 2008, pp. 3–10.
[11] R. Mietzner, T. Unger and F. Leymann, *Cafe: A Generic Configurable Customizable Composite Cloud Application Framework*, On the Move to Meaningful Internet Systems (OTM 2009), 2009, pp. 357–364.
[12] T. Binz, G. Breiter, F. Leyman, and T. Spatzier, *Portable Cloud Services Using TOSCA*, IEEE Internet Computing Magazine 16 (2012), pp. 80–85.
[13] *OpenStack Cloud Software*, http://www.openstack.org/. Last accessed: June 2013.
[14] *Juju* homepage, https://juju.ubuntu.com/. Last accessed: December 2013.
[15] *Amazon Elastic, Compute Cloud*, http://aws.amazon.com/ec2/. Last accessed: June 2013.
[16] *Wordpress*, http://wordpress.org/. Last accessed: July 2013.
[17] *Puppet*, http://puppetlabs.com/. Last accessed: July 2013.
[18] *Chef*, http://www.opscode.com/chef/. Last accessed: 2013.
[19] *MediaWiki*, http://www.mediawiki.org/wiki/MediaWiki. Last accessed: December 2013.
[20] *Memcached*, http://memcached.org/. Last accessed: December 2013.
[21] *HAProxy*, http://haproxy.1wt.eu/. Last accessed: December 2013.
[22] J. Povedano-Molina, L. Foschini, A. Corradi, J. M. Lopez-Vega and J. M. Lopez-Soler, *DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant clouds*, Future Generation Computer Systems, 29 (2013), pp. 2041–2056.

Edited by: Maria Fazio and Nik Bessis
Received: Nov 2, 2013
Accepted: Jan 10, 2014

DOI 10.12694/scpe.v14i4.931
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. 249–263. http://www.scpe.org

A SESSION INITIATION PROTOCOL FOR THE INTERNET OF THINGS *

SIMONE CIRANI, MARCO PICONE AND LUCA VELTRI†

Abstract. The Internet of Things (IoT) refers to the interconnection of billions of constrained devices, denoted as "smart objects", in an Internet-like structure. Smart objects typically feature limited capabilities in terms of computation and memory and operate in constrained environments, such as low-power lossy networks. As the Internet Protocol (IP) has been foreseen as the standard for communications in IoT, an effort to bring IP connectivity to smart objects and define suitable communication protocols (i.e. Constrained Application Protocol (CoAP)) is being carried out within standardization organizations, such as the Internet Engineering Task Force (IETF). In this paper, we propose a constrained version of the Session Initiation Protocol (SIP), named "CoSIP", whose intent is to allow constrained devices to instantiate communication sessions in a lightweight and standard fashion. Session instantiation can include a negotiation phase of some parameters which will be used for all subsequent communication. CoSIP can be adopted in several application scenarios, such as service discovery and publish/subscribe applications, which are detailed. An evaluation of the proposed protocol is also presented, based on a Java implementation of CoSIP, to show the benefits that its adoption can bring about, in terms of compression rate with the existing SIP protocol and message overhead compared with the use of CoAP.

Key words: Internet of Things, communication protocols, CoAP, SIP, signaling, service discovery

1. Introduction. The Internet of Things (IoT) refers to the interconnection of billions of constrained devices, denoted as "smart objects", in an Internet-like structure. Smart objects have limited capabilities, in terms of computational power and memory (e.g., 8-bit microcontrollers with small amounts of ROM and RAM), and might be battery-powered devices, thus raising the need to adopt particularly energy efficient technologies. Smart objects typically operate in constrained networks which often have high packet error rates and a throughput of tens of kbit/s. In order to interconnect smart objects in an Internet-like architecture, standard and interoperable communication mechanisms and protocols are required, and a lot of work is currently ongoing for defining proper standards. It is a common opinion that in the near future IP will be the base common network protocol for the IoT. This does not imply that all objects will be able to run IP. In contrast, there will always be tiny devices, such as tiny sensors or Radio-Frequency IDentification (RFID) tags, that will be organized in closed networks implementing very simple and application-specific communication protocols and that eventually will be connected to an external network through a proper gateway. However, it is foreseen that all remaining small networked objects will exploit the benefits of IP and corresponding protocol suite.

In [1], the author tries to define the following pair of classes for constrained devices, in terms of memory capacity, in order to be used as a rough indication of device capabilities:

- class 1: RAM size = ∼10 KB, Flash size = ∼100 KB
- class 2: RAM size = ∼50 KB, Flash size = ∼250 KB

Some of these networked objects, with enough memory, computational power, and power capacity, will simply run existing IP-based protocol suite implementations. Some others will still run standard Internet protocols, but may benefit from specific implementations that try to achieve better performance in terms of memory size, computational power, and power consumption. Instead, in other constrained networked scenarios, smart objects may require additional protocols and some protocol adaptations in order to optimize Internet communications and lower memory, computational, and power requirements.

As billions of smart objects are expected to come to life and IPv4 addresses have eventually reached depletion, IPv6 [2] has been identified as a candidate for smart-object communication at the network layer. However, due to the possible limitations that constrained devices may encounter, some adaptation at network layer (IP) and at upper layers may be required.

*The work of Simone Cirani and Luca Veltri has been funded by the European Community's Seventh Framework Programme, area "Internetconnected Objects", under Grant no. 288879, CALIPSO project - Connect All IP-based Smart Objects. The work reflects only the authors views; the European Community is not liable for any use that may be made of the information contained herein.

†Department of Information Engineering, University of Parma, Viale G.P. Usberti, 181/A, 43124 Parma, Italy (simone.cirani, marco.picone, luca.veltri@unipr.it).

Significant reasons for proper protocol optimizations and adaptations for resource-constrained objects can be summarized as follows.

- Smart objects typically use, at physical and link layers, communication protocols (such as IEEE 802.15.4) which are characterized by small Maximum Transmission Units (MTUs), thus leading to packet fragmentation. In this case, the use of compressed protocols can significantly reduce the need for packet fragmentation and postponed transmissions.
- Processing larger packets likely leads to higher energy consumption, which can be a critical issue in battery-powered devices.
- Minimized versions of protocols (at all layers) can reduce the number of exchanged messages.

For this reason, within the IETF some specific working groups have been set. In particular, the IETF 6LoWPAN (IPv6 over Low power WPAN) Working Group [3] is defining encapsulation and other adaptation mechanisms to allow IPv6 packets to be sent to and received from over Low power Wireless Personal Area Networks, such as those based on IEEE 802.15.4. for the network layer, the ROLL (Routing Over Low power and Lossy networks) Working Group [4] is currently studying and defining proper routing mechanisms. Instead, for the application layer, the IETF CoRE (Constrained RESTful Environments) Working Group [5] is currently defining a Constrained Application Protocol (CoAP) [6], to be used as a generic web protocol for constrained environments, targeting Machine-to-Machine (M2M) applications, and that can be seen in some ways as a compressed version of the HyperText Transfer Protocol (HTTP) [7]. CoAP will include the following features:

- request/response interaction model between application endpoints;
- built-in discovery of services and resources;
- key concepts of the Web such as URIs (Uniform Resource Identifiers) and Internet media types.

The typical Internet of Things protocol stack, compared with the standard web protocol stack, is depicted in Fig. 1.1. The symmetry between the the two protocol stacks is clear: for instance, at the application layer, while HTTP is the most widespread application protocol for Internet applications, such as the World Wide Web, its constrained version, the CoAP, is expected to be used, reducing the complexity of implementation as well as the size of packets exchanged. CoAP, in contrast to HTTP, uses UDP [8] as a lightweight transport protocol.

CoAP is intended to provide application a RESTful (Representational state transfer) communication mechanism. According to the REST model, representations of resources are exchanged between a client and a server. A resource representation is the current or intended state of a resource referred to the server through a proper namespace. A client that is interested in the state of a resource sends a request to the server; the server then responds with the current representation of the resource. An example of CoAP request/response interaction is depicted in Fig. 1.2.

If the client is interested in receiving the representation during a period of time, according to this model the client should periodically repeat such requests in order to always have the updated representation of the resource. Of course this mode of operation would lead to unnecessary messages sent over the network and processing load at both client and server sides, each time a request is sent for a resource state that is not changed. In order to save both network and processing resources, a more suitable communication model, following the event subscribe/notify paradigm, can be used. According to this paradigm, a client, called "observer", interested in a resource subscribes to a resource, called "subject", informing the server that it is interested in being notified whenever the subject changes the state; we say that the clients want to "observe" the resource.

There is a CoAP extension [9] currently in the IETF standardization process, that aims at defining a CoAP-based subscribe/notify service can be implemented over the basic CoAP REST model. Figure 1.3 compares the two approaches of "observing" a resource.

However, beside REST and subscribe/notify service models, there are also many other applications in both constrained and non-constrained environments that feature non-request/response communication models. Some of these applications require the creation and management of a "session". With the term of "session" we refer to any exchange of data between an association of participants. In case of two participants, the session may involve the sending of one or (probably) more data packets from one participant to the other, in one or both directions. In case of unidirectional sessions, they may be initiated by both the sender or the receiver. Examples of sessions in IoT scenario may be the data flow generated by a sensor (measurement samples) and sent to a

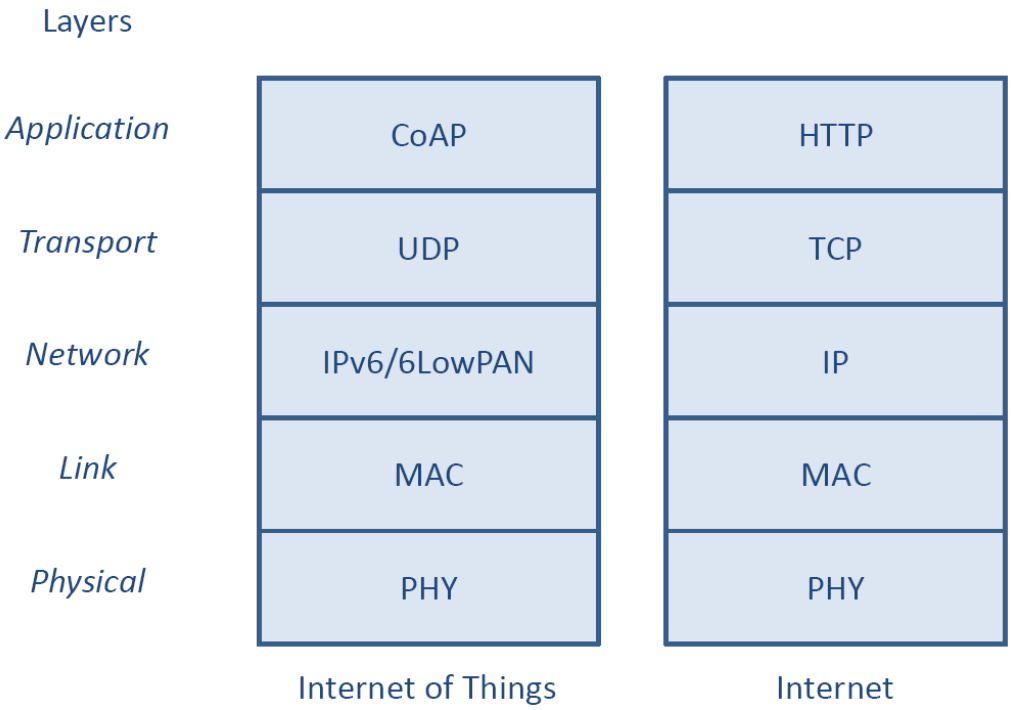


FIG. 1.1. *Comparison between the IoT and the Internet protocol stack for OSI layers.*

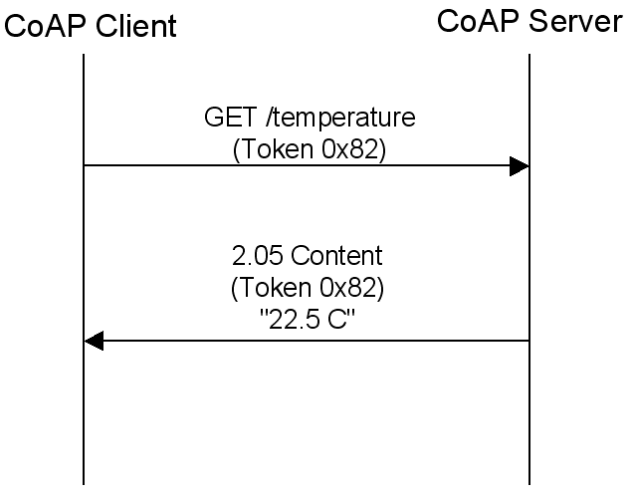


FIG. 1.2. *CoAP request/response model.*

given recipient for further processing, or some data streams exchanged by two interacting toys.

Although in principle CoAP encapsulation could be used also for carrying data in non-request/response fashion, for example by using CoAP POST request in non-confirmable mode, or by using the CoAP "observation" model, it is evident that could be much more efficient to setup a session between constrained nodes first, and then perform a more lightweight communication without carrying unnecessary CoAP header fields for each data packet. The data communication will be in accord to the network, transport, and application parameters negotiated during the session setup.

The Session Initiation Protocol (SIP) [10] is the standard application protocol for establishing application-

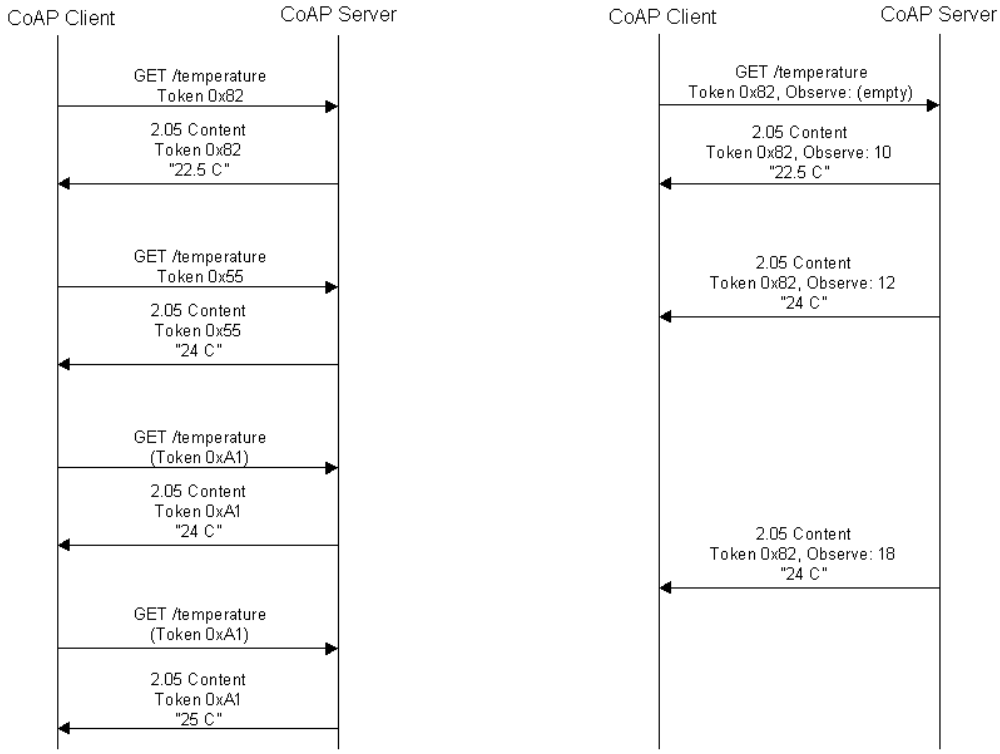


FIG. 1.3. *CoAP basic model (left) vs. CoAP "observing" (right). The CoAP "observe" extension introduces a subscribe/notify communication model, in contrast to the request/response model of standard CoAP.*

level sessions. It allows the endpoints to create, modify, and terminate any kind of (multi)media sessions, such as VoIP calls, multimedia conferences, or data communications. Once a session has been established, the media are transmitted typically by using other application-layer protocols, such as RTP and RTCP [11], or as raw UDP data, directly between the endpoints, in a peer-to-peer fashion. SIP is a text protocol, similar to HTTP, which can run on top of several transport protocols, such as UDP (default), TCP, or SCTP, or on top of secure transport protocol such as TLS and DTLS. Session parameters are exchanged as SIP message payloads; a standard protocol used for this purpose is the Session Description Protocol [12]. The SIP protocol also supports intermediate network elements which are used to allow endpoint registration and session establishment, such as SIP Proxy servers and Registrar servers. SIP also defines the concepts of transaction, dialog, and call as groups of related messages, at different abstraction layers.

Although SIP has been defined for Internet application, we may think to re-use it also in constrained IoT scenario. Note that SIP already includes mechanisms for subscribe/notify communication paradigms [13] and for resource directory, particularly useful in IoT scenarios, for which proper CoAP extensions are currently being specified [9, 14].

The main drawback of using standard SIP protocol in constrained environments is the large size of text-based SIP messages (compared to other binary protocols such CoAP), and the processing load required for parsing such messages.

For this reason, in this paper, we propose a constrained version of the Session Initiation Protocol, named "CoSIP", whose intent is to allow constrained devices to instantiate communication sessions in a lightweight and standard fashion and can be adopted in M2M application scenarios. Session instantiation can include a negotiation phase of some parameters which will be used for all subsequent communication. The proposed CoSIP is a binary protocol which maps to SIP, similarly to CoAP does to HTTP. CoSIP can be adopted in several application scenarios, such as service discovery and publish/subscribe applications.

The rest of this paper is organized as follows. In Section 2, an overview of related works is presented. In Section 3, fundamentals of the SIP protocol are summarized. In Section 4, the proposed CoSIP protocol is detailed together with its architecture and preliminary implementation. Some use cases for CoSIP-based applications in Internet of Things scenarios are presented in Section 5 and a performance evaluation of the proposed protocol is shown in Section 6. Finally, in Section 7 we draw our conclusions.

2. Related Work. Smart objects typically are required to operate using low-power and low-rate communication means, featuring unstable (lossy) links, such as IEEE 802.15.4, usually termed Low-power Wireless Personal Area Networks (LoWPANs) or Low-power and Lossy Networks (LLNs). The Internet Engineering Task Force (IETF) has setup several working groups in order to address many issues related to bringing IP connectivity to LoWPAN smart objects. In particular, the 6LoWPAN (IPv6 over Low power WPAN) WG [3] was chartered to work on defining mechanisms that optimize the adoption of IPv6 in LoWPANs and the ROLL (Routing Over Low power and Lossy networks) WG [4] was chartered to develop optimal IPv6 routing in LLNs. Finally, the CoRE (Constrained RESTful Environments) WG [5] has been chartered to provide a framework for RESTful applications in constrained IP networks. The CoRE WG is working on the definition of a standard application-level protocol, named CoAP, which can be used to let constrained devices communicate with any node, either on the same network or on the Internet, and provides a mapping to HTTP REST APIs. CoAP is intended to provide, among others, Create-Read-Update-Delete (CRUD) primitives for resources of constrained devices and publish/subscribe communication capabilities. While the work on CoAP is already at an advanced stage, the CoRE WG is also investigating mechanisms for discovery and configuration, but the work on these issues is still at an early stage and therefore open to proposals.

The "observer" CoAP extension [9] allows CoAP clients to observe resources (subscribe/notify mechanism) and be notified when the state of the observed resource changes. This approach requires the introduction of a new CoAP *Observe* option to be used in GET requests in order to let the client register its interest in the resource. The server will then send "unsolicited" responses back to the client echoing the token specified by the client in the GET request and reporting an Observe option with a sequence number used for reordering purposes. As we will describe later, we envision that the instantiation of a session could significantly reduce the amount of transmitted bytes, since, after the session has been established, only the payloads could be sent to the observer, thus eliminating the overhead due to the inclusion of the CoAP headers in each notification message.

As for service discovery, the CoRE WG has defined a mechanism, denoted as *Resource Directory* (RD) [14], to be adopted in M2M applications. The use of a RD is necessary because of the impracticality of a direct resource discovery, due to the presence of duty-cycled nodes and unstable links in LLNs. Each CoAP server must expose an interface */.well-known/core* to which a client can send requests for discovering available resources. The CoAP server will reply with the list of resources and, for each resource, an attribute that specifies the format of the data associated to that resource. The CoAP protocol, however, does not specify how a node joining the network for the first time must behave in order to announce itself to the resource directory node. In [15], this functionality is extended to multicast communications. In particular, multicast Resource Discovery is useful when a client needs to locate a resource within a limited scope, and that scope supports IP multicast. A *GET* request to the appropriate multicast address is made for */.well-known/core*. Of course this multicast Resource Discovery works only within an IP multicast domain and does not scale to larger networks that do not support end-to-end multicast.

The registration of a resource in the RD is performed by sending a POST request to the RD, while the discovery can be accomplished by issuing a GET request to the RD targeting the .well-known/core URI. This discovery mechanism is totally self-contained in CoAP as it uses only CoAP messages.

The adoption of the CoSIP protocol provides an alternative mechanism to register resources on a RD, which may be also called CoSIP Registrar Server. The advantage of using a CoSIP based registration mechanism is that it might be possible to register resources other than those reachable through CoAP, thus providing a scalable and generic mechanism for service discovery in constrained applications with a higher degree of expressiveness, such as setting an expiration time for the registration.

3. SIP. The Session Initiation Protocol (SIP) [10] is an IETF standard application-layer control protocol that can be used to establish, modify, or terminate end-to-end sessions. SIP is a text-based client-server protocol,

where the client sends SIP requests and the server responds to requests. SIP architecture includes both end systems (terminals), also called SIP user agents, and intermediate systems, called SIP proxy, redirect or registrar servers, depending on their function. A "registrar server" is SIP server that receives registration requests issued by SIP user agents, and used for maintaining the binding between the SIP user name also called address-of-record SIP AOR, and its current contact address, that can be used for reaching such user/resource. The mapping between SIP AORs and SIP contact URIs is called Location Service and is an important component for resource discovery in SIP.

All SIP addresses are represented by URIs with the scheme "sip:", and identify a name or contact address of a SIP user; a SIP user can be a real user, a system, an application, or a any kind of resources.

The proxy servers are intermediary entities that act as both server and client for making requests on behalf of other clients. A proxy server may act as "outbound" proxy when used for routing SIP request addressed to a user that is not maintained in a local Location Service, or as "far-end" (or "destination") proxy if the request is addressed to a user with an AOR maintained by the proxy and mapped to one or more SIP contact URIs.

Differently by the proxy servers, the redirect servers accept requests and replies to the client with a response message that typically provides a different contact address (URI) for the target of previous request.

SIP signaling between users consists of requests and responses. When a UA wants to send a request to a remote user (identified by a SIP AOR), it may send the message directly to the IP address of the remote user's UA, or to the proxy server that is responsible for the target AOR (normally the fully qualified domain name (FQDN) of the proxy server is included in the AOR), or to a locally configured outbound proxy server. When the request reaches the target UA, the latter may optionally replies with some provisional 1xx responses and with one final response (codes 2xx for success, 3xx, 4xx, 5xx and 6xx for failure).

SIP defines different request methods such as INVITE, ACK, BYE, CANCEL, OPTIONS, REGISTER, SUBSCRIBE, NOTIFY, atc.

When a UA wants to initiate a session it sends an INVITE message that may be responded with provisional 1xx responses and a final response. The UA that issued the INVITE then have to confirm the final response with a ACK message. Differently by all other SIP transaction, the INVITE transaction is a three-way handshake (INVITE/2xx/ACK).

Once the session is established, both endpoints (user agents) may modify the session with a new INVITE transaction, or tear-down the session with a BYE transaction (BYE/2xx). When the caller or the callee wish to terminate a call, they send a BYE request. SIP messages may contain a "body" that is treated as opaque payload by SIP.

Figure 3.1 shows an example of SIP message flow, including the registration of two UAs with their own registrar/proxy servers, and a session setup and tear-down from UA1 (identified by the SIP AOR sip:u1@P1) to UA2 (identified by the SIP AOR sip:u2@P2).

During an INVITE transaction the SIP body is used to negotiate the session in terms of transport and application protocol, IP addresses and port number, payload formats, encryption algorithms and parameters, etc. The negotiation follows an offer/answer paradigm, where the offer is usually sent within the INVITE while the answer is in the 2xx final response. The most used protocol for such negotiation is the Session description Protocol (SDP); however other prtocol may be used.

4. CoSIP. As described in Section 1, in both constrained and non-constrained environments there are many applications that may require or simply may obtain advantages by negotiating end-to-end data sessions. In this case the communication model consists in a first phase in which one endpoint requests the establishment of a data communication and, optionally, both endpoints negotiate some communication parameters (transfer protocols, data formats, endpoint IP addresses and ports, encryption algorithms and keying materials, and other application specific parameters) of the subsequent data sessions. This may be useful for both client-server or peer-to-peer applications, regardless the data sessions evolve or not according to a request/response model. The main advantage is that all such parameters, including possible resource addressing, may be exchanged in advance, while no such control information is required during data transfer. The longer the data sessions, the more the advantage is evident respect to a per-message control information. Also in the case of data sessions that may vary formats or other parameters during time, such adaptation may be supported by performing session renegotiation. A standard way to achieve all this onto an IP-based network may be by using the Session

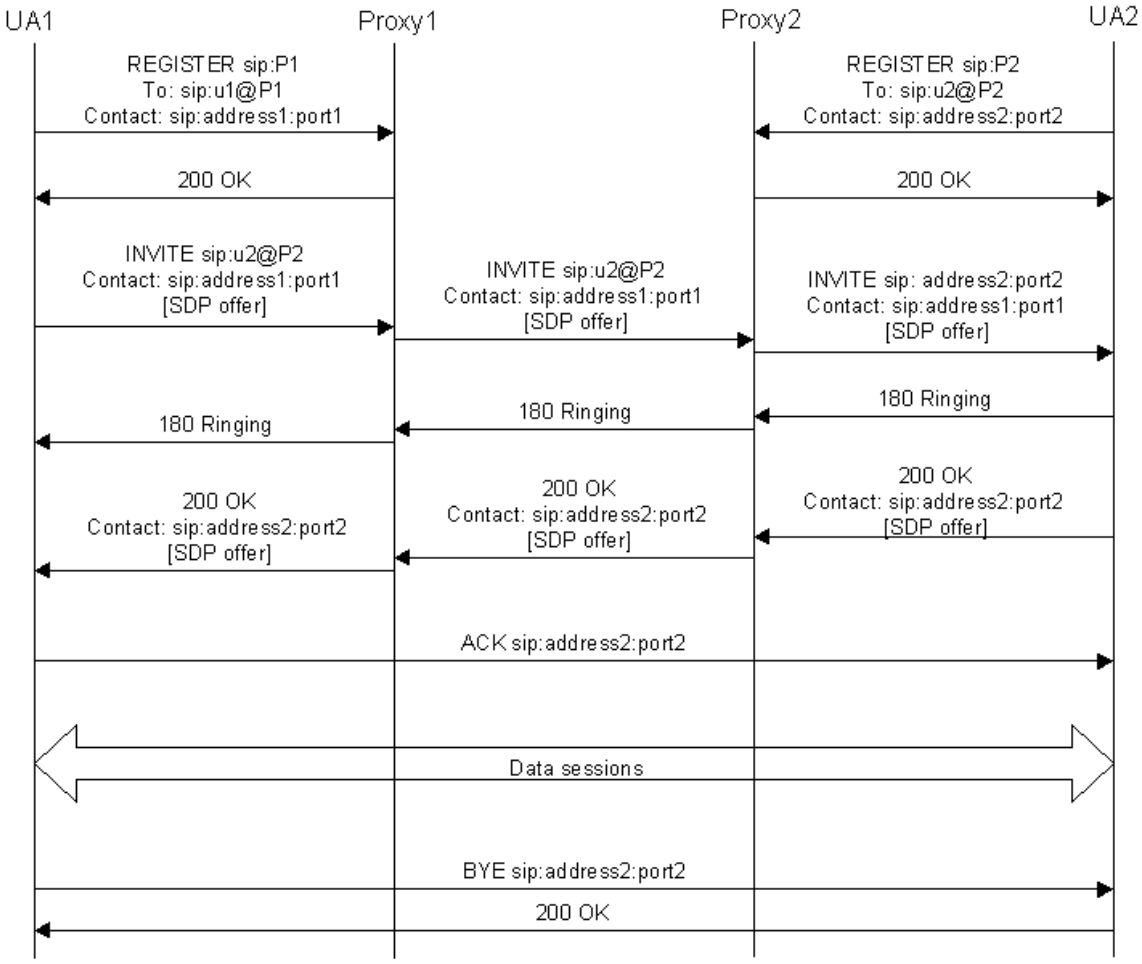


Fig. 3.1. *UA registrations and session setup with two intermediate proxy servers.*

Initiation Protocol [10]. In fact SIP has been defined as standard protocol for initiating, modifying and tearing down any type of end-to-end multimedia sessions. SIP is independent from the protocol used for data transfer and from the protocol used for negotiating the data transfer (such negotiation protocol can be encapsulated transparently within the SIP exchange). In order to simplify the implementation, SIP reuses the same message format and protocol fields of HTTP. However, in contrast to HTTP, SIP works by default onto UDP, by directly implementing all mechanisms for a reliable transaction-based message transfer. This is an advantage in duty-cycled constrained environment where some problems may arise when trying to use connection-oriented transports, such as TCP. However, SIP may also run onto other transport protocols such as TCP, SCTP, TLS or DTLS. Unfortunately SIP derives from HTTP the text-based protocol syntax that, even if it simplifies the implementation and debugging, results in i) larger message sizes, and ii) bigger processing costs and probably larger source code size (RAM footprint) required for message parsing. Note that the SIP standard defines also a mechanism for reducing the overall size of SIP messages; this is achieved by using a compact form of some common header field names. However, although it allows a partial reduction of the message size, it may still result in big messages, especially if compared to other binary formats, for example those defined for CoAP. For this reason we tried to define and implement a new binary format for SIP in order to take advantages of the functionalities already defined and supported by SIP methods and functions, together with a very compact message encoding. We naturally called such new protocol CoSIP, that stands for Constrained Session Initiation Protocol, or, simply, Constrained SIP. Due to the protocol similarities between SIP and HTTP, in order to

maximize the reuse of protocol definitions and source code implementations, we decide to base CoSIP onto the same message format that has been defined for CoAP, thanks to the role that CoAP plays respect to HTTP. However, it is important to note that, while CoAP required to define new message exchanges, mainly due to the fact that CoAP need to operated in constrained and unreliable networked scenario over UDP transport protocol, while HTTP works over TCP, CoSIP may completely reuse all SIP message exchanges and transactions already defined by the SIP standard, since SIP already works over unreliable transport protocols (e.g. UDP).

SIP is structured as a layered protocol, where at the top there is the concept of dialog, that is a peer-to-peer relationship between two SIP nodes that persists for some time and facilitates sequencing of different request-response exchanges (transactions). In CoAP there is no concept equivalent to SIP dialogs, and, if needed, it has to be explicitly implemented at application level. Under the dialog there is the transaction layer, that is the message exchange that comprises a client request, the following optional server provisional responses and the server final response. The concept of transaction is also present in CoAP where requests and responses are bound and matched through a token present as message header field. Under the transaction there is the messaging layer where messages are effectively formatted and sent through an underlying non-SIP transport protocol (such as UDP or TCP). Instead of completely re-designing a session initiation protocol for constrained environments, we propose to reuse the SIP layered architecture of SIP, by simply re-defining the messaging layer with a constrained-oriented binary encoding. For such a purpose, we propose to reuse the same CoAP message syntax [6]. Figure 4.1 shows the CoSIP message format derived by CoAP. A CoSIP message starts with the 2-bit Version field (set to 1, i.e. CoSIP version 1), followed by the 2-bit Type field (set to 1 = Non-confirmable), the 4-bit CoAP TKL field (set to 0), the 8-bit Code field that encode request methods (for request messages) and response codes (for response messages), the 16-bit CoAP Message ID field, followed by zero or more Option fields. In case a CoSIP message body is present, as in CoAP it is appended after Options field, prefixed by an 1-byte marker (0xFF) that separates CoSIP header and payload. Options are encoded as in CoAP in Type-Length-Value (TLV) format and encode all CoSIP header fields (From, Via, Call-ID, etc.) included in the CoSIP message.

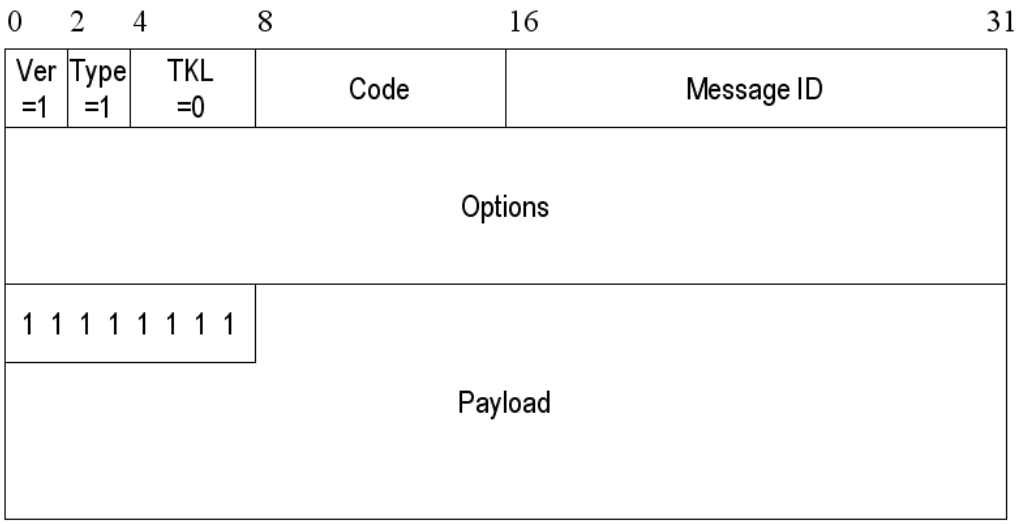


Fig. 4.1. *CoSIP message format.*

Since CoSIP re-uses the transaction layer of SIP, no CoAP optional Token field is needed [6] and the TKL (Token Length) field can be permanently set to 0. Moreover, since CoSIP already has reliable message transmission (within the transaction layer), no Confirmable (0), Acknowledgement (2) nor Reset (3) message types are needed, and the only type of message that must be supported is Non-confirmable (1).

The comparison of the layered architecture of CoSIP and SIP is shown in Fig. 4.2.

Beside the above binary message, a CoSIP message can be virtually seen as a standard SIP message,

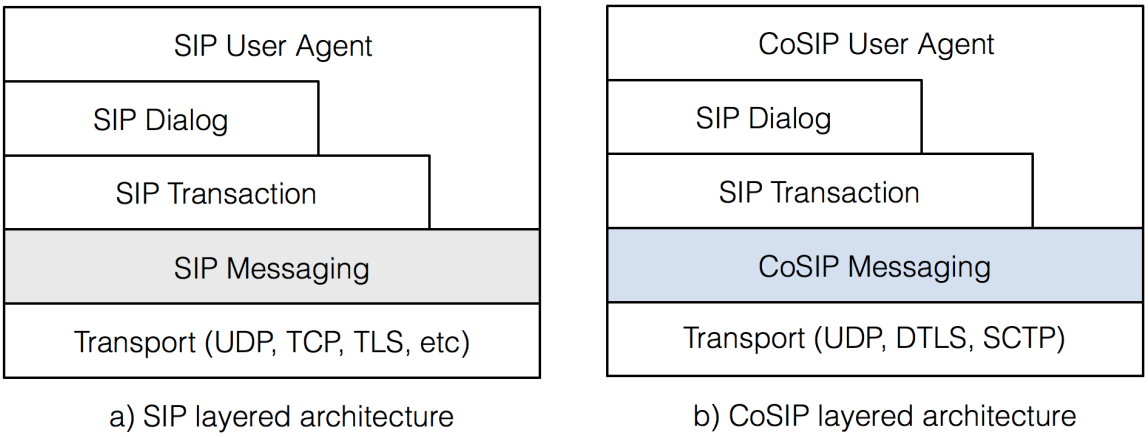


FIG. 4.2. *Comparison of the layered architectures of SIP (a) and CoSIP (b).*

formed by one request–line or one status–line (depending if the message is a request or a response), followed by a sequence of SIP header fields, followed by a message body, if preset. In particular, SIP header fields are logically the same of the standard SIP protocol, properly encoded in corresponding CoSIP Options. For each SIP header field a different option number has been set, and a proper encoding mechanism has been defined. In particular general rules that we followed are:

- IP addresses are encoded as a sequence of 5 bytes for IPv4 and 17 bytes for IPv6, where the first byte discriminates the type of address, i.e. 1 = IPv4 address, 2 = IPv6 address, 3 = FQDN (fully qualified domain name);
- for header field parameters, when possible, the parameter name is implicitly identified by the position of its value in the corresponded binary-encoded CoSIP Option; otherwise parameter names are substituted by parameter codes; in the latter case the parameter is encoded as type-value pair (in case of fixed size values) or type-length-vale tuple (in case of variable size values);
- random tokens, such as SIP "branch" values, SIP "from" and "to" tags, "call-id", etc. are generated as arrays of maximum 6 bytes.

One problem in reusing the current CoAP message format [6] is that in CoAP the 8-bit Code field is used to encode all possible request methods and response codes. In particular, for response messages the 8-bit Code field is divided by CoAP into two sub-fields "class" (3 bits) and "details" (5 bits); the upper three bits ("class") encodes the CoAP response classes 2xx (Success), 4xx (Client Error), and 5 (Server Error), while the remaining 5 bits ("details") encode the sub-type of the response within a given class type. For example a 403 "Forbidden" response is encoded as 4 ("class") and 03 ("details"). Unfortunately, this method limits the number of possible response codes that can be used (for example, using only 5 bits for "details" does not allow the direct encoding of response codes such as 480 "Temporarily Unavailable" or 488 "Not Acceptable Here"). In CoSIP, we overcome this problem by encoding within the Code field only the response class (2xx, 4xx, etc.) and by adding an explicit Option field, called "Response-Code" option, that encodes the complete response code (e.g. 488), including the response sub-type (88, in the case of response code 488). The size of the "Response-Code" option is 2 bytes. Moreover, in order to support all SIP/CoSIP response codes we also added the classes 1xx (Provisional) and 3xx (Redirect) used in SIP.

5. IoT Application Scenarios. In this section, we will describe the most significant for IoT applications, intended to provide an overview of the capabilities and typical usage of the CoSIP protocol. In all the scenarios, we consider a network element, denoted as "IoT Gateway", which includes also a HTTP/CoAP proxy, which can be used by nodes residing outside the constrained network to access CoAP services.

5.1. CoAP Service Discovery. CoSIP allows smart objects to register the services they provide to populate a CoSIP Registrar Server, which serves as a Resource Directory. The terms "Registrar Server" and

"Resource Directory" are here interchangeable.

Figure 5.1 shows a complete service registration and discovery scenario enabled by CoSIP. We consider a smart object that includes a CoAP server, which provides one or more RESTful services, and a CoSIP agent, which is used to interact with the CoSIP Registrar Server. The smart object issues a `REGISTER` request (denoted with the letter "a" in the figure) which includes registration parameters, such as the Address of Record (AoR) of the CoAP service and the actual URL that can be used to access the resource (Contact Address). Note that, while the original SIP specification states that the To header MUST report a SIP or SIPS URI, CoSIP allows to specify any scheme URI in the To header, e.g. a CoAP URI. Upon receiving the registration request, the Registrar Server stores the AoR-to-Contact Address mapping in a Location Database and then sends a `200 OK` response.

When a REST client, either CoAP or HTTP, is willing to discover the services, it can issue a `GET` request targeting the .well-known/core URI, which is used as a default entry point to retrieve the resources hosted by the Resource Directory, as defined in [16]. The `GET` request is sent to the HTTP/CoAP proxy, which returns a `200 OK` (in the case of HTTP) or a `2.05 Content` (in the case of CoAP) response containing the list of services in the payload.

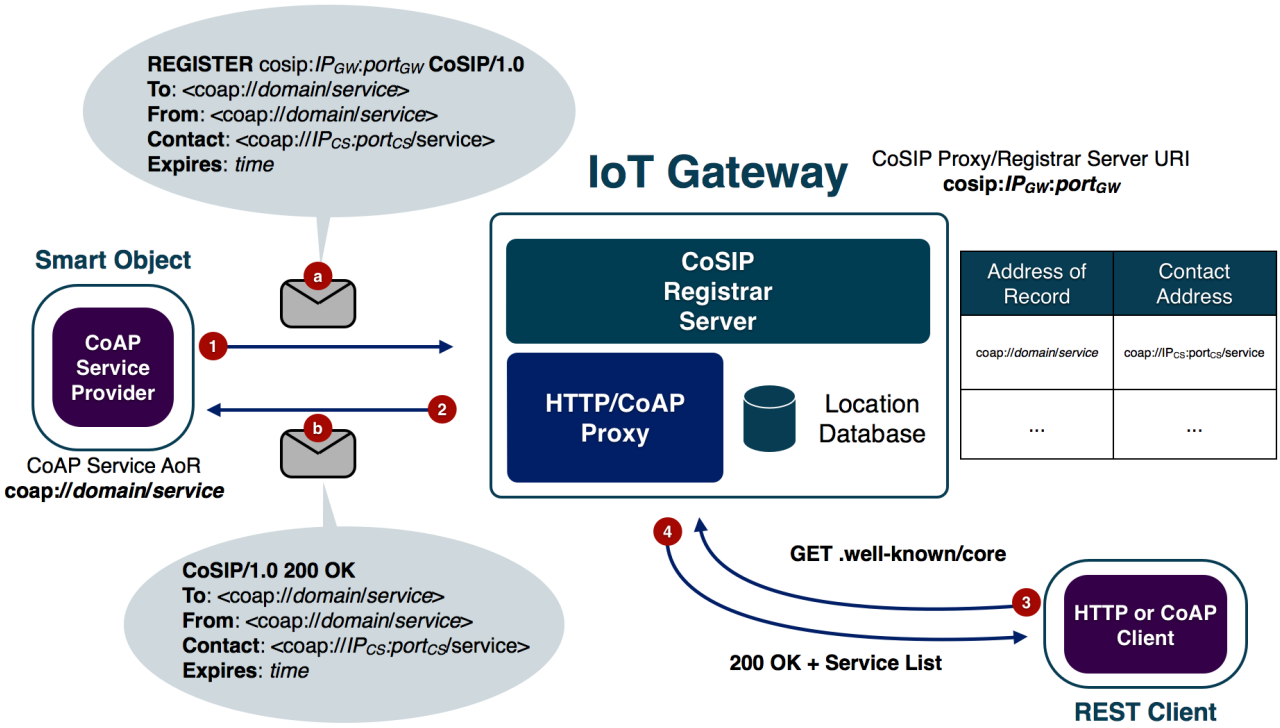


FIG. 5.1. *CoAP Service Discovery.*

5.2. Session Establishment. A session is established when two endpoints need to exchange data. CoSIP allows the establishment of session in a standard way without binding the session establishment method to a specific session protocol. For instance, CoSIP can be used to negotiate and instantiate a RTP session between constrained nodes. Once a session has been established, the data exchange between the endpoints occurs (logically) in a peer-to-peer fashion.

Figure 5.2 shows how CoSIP can be used to establish a session between two endpoints. Let's assume an IoT Agent (IoT-A_1) identified by the CoSIP URI *cosip:user1@domain*, which includes at least a CoSIP agent, has registered its contact address to an IoT Gateway in the same way as described in the previous subsection (steps 1 and 2). If another IoT-A_2 *cosip:user2@domain* wants to establish a session with IoT-A_1, it will send a proper `INVITE` request to the IoT Gateway, which will act as a CoSIP Proxy relaying the request to IoT-A_1

(steps 3 and 4). IoT-A_1 will then send a `200 OK` response to IoT-A_2 (steps 5 and 6), which will finalize the session creation by sending an `ACK` message to IoT-A_2 (steps 7 and 8).

At this point the session has been setup and data flow between IoT-A_1 and IoT-A_2 can occur directly. The session establishment process can be used to negotiate some communication parameters, for instance by encapsulating Session Description Protocol (SDP) [12] or equivalent in the message payload. As we will show in the evaluation section, setting up a session, rather than using CoAP, both in a request/response or subscribe/notify paradigm, is a very efficient approach to avoid the burden of the overhead due to carrying headers in each exchanged message since eventually only the payloads would be relevant for the application.

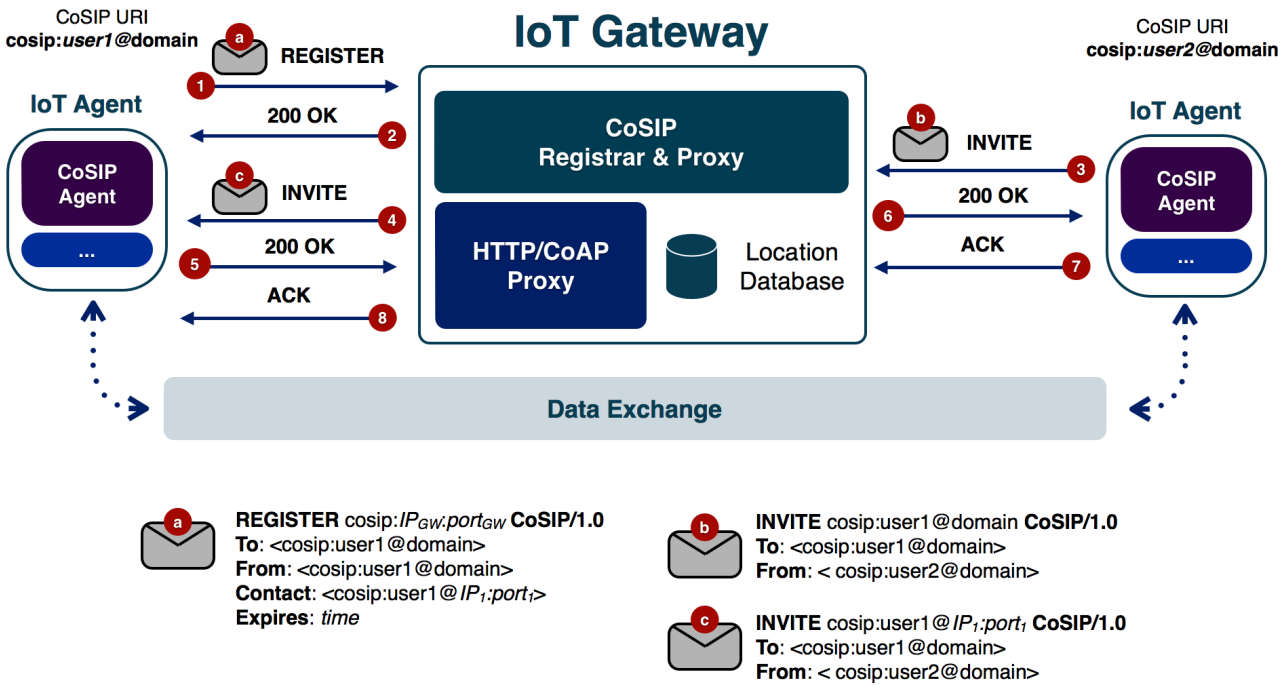


FIG. 5.2. *CoSIP Session Establishment.*

5.3. Subscribe/Notify Applications. IoT scenarios typically involve smart objects which might be battery-powered devices. It is crucial to adopt energy-efficient paradigms, e.g. OS tasks, application processing, and communication. In order to minimize the power consumed, duty-cycled smart objects are adopted. Sleepy nodes, especially those operating in LLNs, aren't guaranteed to be reached, therefore it is more appropriate for smart objects to use a Subscribe/Notify, also denoted as Publish/Subscribe (Pub/Sub), approach to send notifications regarding the state of their resources, rather than receive and serve incoming requests. Such a behavior can be achieved by leveraging on the inherent capabilities of SIP, and therefore of CoSIP, as sketched in Fig. 5.3.

The depicted scenarios considers several Pub/Sub interactions: notifications can be sent either by a Notifier IoT Agent (IoT-A_N) or by an IoT Gateway, and subscribers can be either Subscriber IoT Agents (IoT-A_S), IoT Gateways, or generic Remote Subscribers. Let's assume that all the notifiers have previously registered with their CoSIP Registrar Server (this step is also denoted as the Publishing phase in a typical Pub/Sub scenario). The standard subscription/notification procedure is the following:

1. the subscriber sends a `SUBSCRIBE` request to the notifier, also specifying the service events it is interested in;
2. the notifier stores the subscriber's URI and event information and sends a `200 OK` response to the subscriber;
3. whenever the notifier's state changes, it sends a `NOTIFY` request to the subscriber;

4. the subscriber sends a `200 OK` response back to the notifier.

Figure 5.3 reports all the use cases when a Pub/Sub might be used. An IoT-A_S can subscribe to the service of an IoT-A_N in the same network, in case it is willing to perform some task, such as data/service aggregation. The IoT Gateway can subscribe to the IoT-A_N's in order to collect sensed data, e.g. to store them in the cloud, without the need to periodically poll for data. Finally, the IoT Gateway itself might be a notifier for remote subscribers, which are interested in notifications for specific services provided by the gateway, which may or may not be the same of existing IoT-A_N nodes managed by the gateway. Note that, it might be possible to have interaction with legacy SIP agents in case the IoT Gateway is also able to perform SIP/CoSIP proxying.

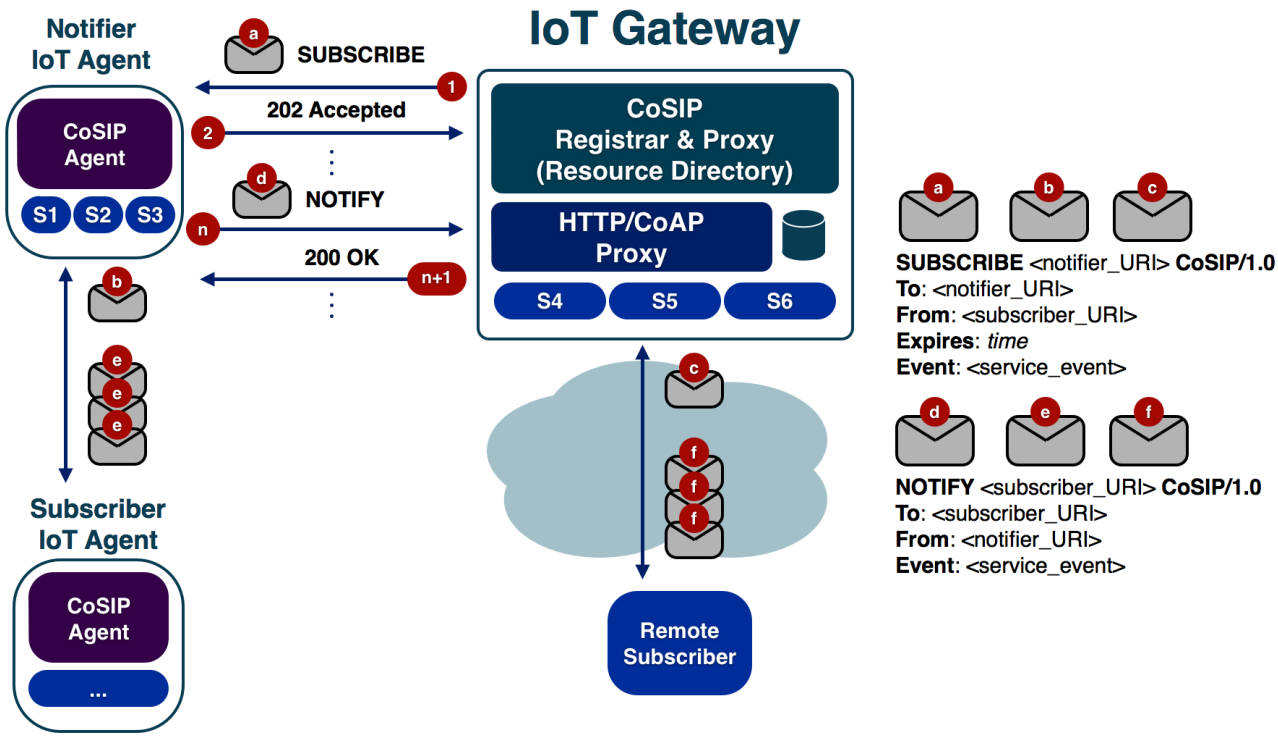


FIG. 5.3. *Subscribe/Notify applications with CoSIP.*

The adoption of CoSIP in IoT scenarios allows to easily set up efficient Pub/Sub-based applications in a standard way, thus allowing for seamless integration and interaction with the Internet. Moreover, the valuable experience gained in the past years with SIP, both in terms of technologies and implementations, can be reused to speed up the implementation and deployment of session-based applications.

6. Protocol Evaluation. In order to evaluate the performance of CoSIP, an implementation of the protocol has been developed together with some test applications. In this work, we have decided to focus on network performance as a metric by measuring the amount of network traffic generated by the test applications. The CoSIP protocol has been implemented in Java language, due to its simplicity, cross-platform support, and the availability of already developed SIP and CoAP libraries [17, 18]. The source code of the CoSIP implementation is freely available at [19].

The performance results show that many advantages can be achieved by using CoSIP, both in constrained and non-constrained applications. The first evaluation compares CoSIP and SIP in terms of bytes transmitted for the signaling part related to the instantiation and termination of a session. Each CoSIP request and response message is separately compared with its SIP counterpart. The results are illustrated in Fig. 6.1. Table 6.1 shows the compression ratio for each CoSIP/SIP message pair. Regarding the session as a whole, CoSIP yields an overall compression ratio of slightly more than 0.55.

Message type	CoSIP (bytes)	SIP (bytes)	compression ratio
INVITE	311	579	0.537
100 Trying	141	279	0.505
180 Ringing	173	372	0.465
200 OK	293	508	0.577
ACK	216	363	0.595
BYE	183	309	0.592
200 OK	162	274	0.591

TABLE 6.1
Comparison between CoSIP and SIP signaling (bytes per message) for session instantiation and establishment.

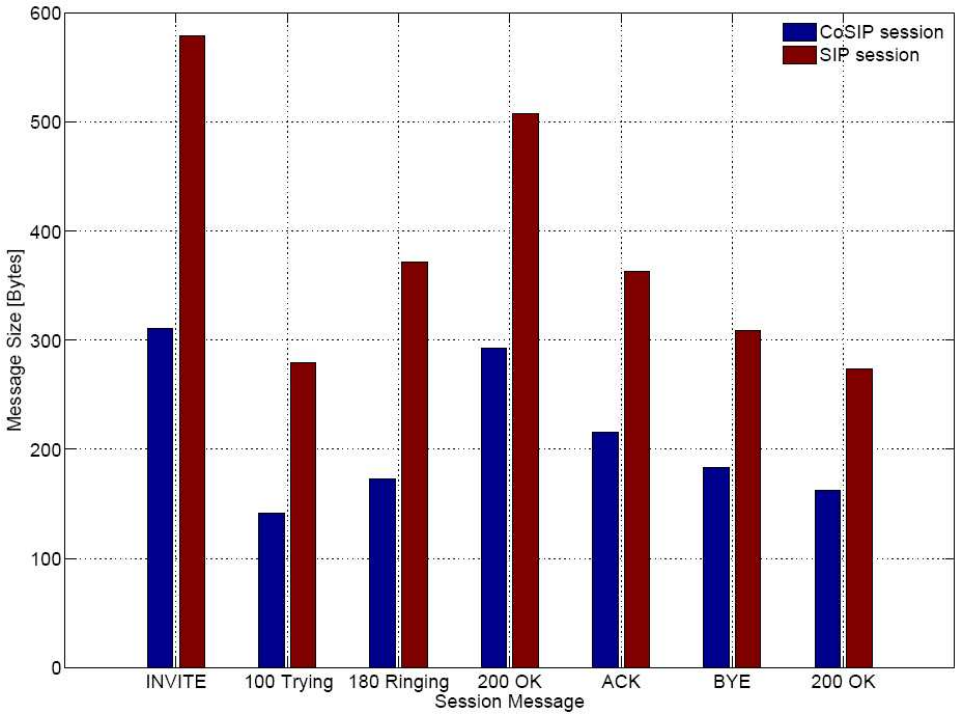


FIG. 6.1. *Transmitted bytes for CoSIP and SIP session (signaling only).*

Another evaluation has been made to show the advantage of using session in constrained applications. Figure 6.2 shows the amount of network traffic (in bytes) generated by two constrained applications: the first application uses CoSIP to establish a session and then performs the data exchange by sending the payloads over UDP; the second is a standard CoAP-based application where the communication occurs between a CoAP client and a CoAP server, using confirmed CoAP POST requests. In both cases data is sent at the same rate of one data message every 2 seconds. The figure shows that the lightweight CoSIP session is instantiated in a very short period of time and after the session has been established few bytes are exchanged between the endpoints. On the other hand the CoAP-based application has no overhead at the beginning due to the instantiation of the session but, soon after, the amount of traffic generated by this application exceeds that of the CoSIP-based application, since in the CoAP-based scenario data is exchanged within CoAP messages resulting in an unnecessary CoAP overhead.

Note that in the depicted scenario the CoSIP signaling used for session initiation includes all SIP header fields normally used in standard non-constrained SIP application, that is no reduction in term of header fields has been performed. Instead for the CoAP application we considered only mandatory CoAP header fields

resulting in the best-case scenario for CoAP in term of CoAP overhead (minimum overhead). This means that in other CoAP applications the slope of the line could become even steeper, thus reducing the time when the break-even point with CoSIP is reached.

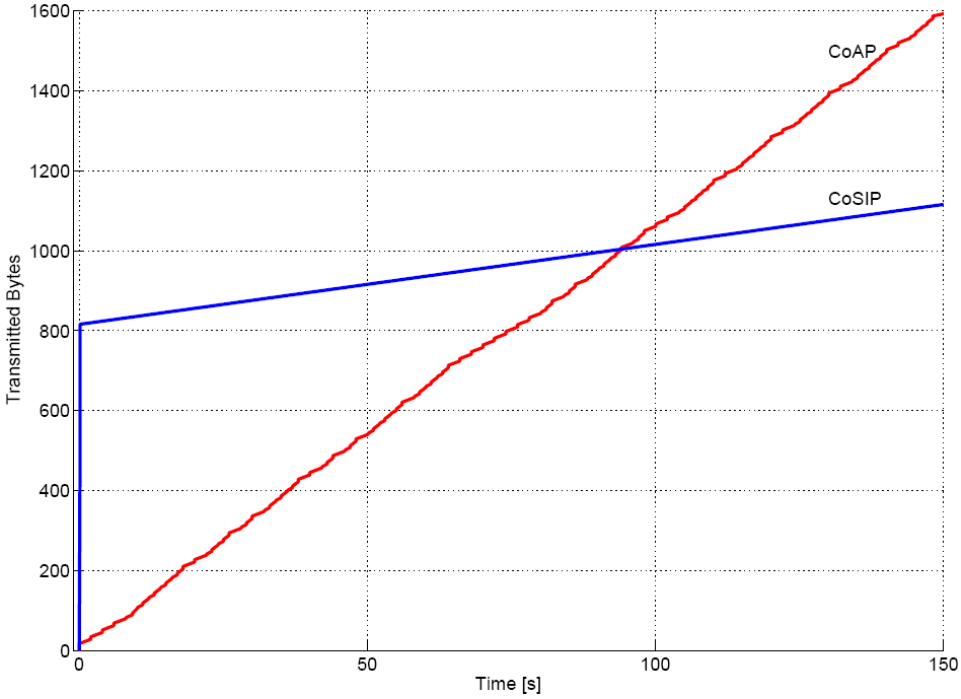


FIG. 6.2. *Transmitted bytes in a CoSIP Session vs. CoAP confirmed POST requests and responses.*

7. Conclusions. In this paper, we have introduced a low-power protocol, named "CoSIP", for establishing sessions between two or more endpoints targeting constrained environments. Many applications, both in constrained and non-constrained scenarios, do benefit from establishing a session between the participants in order to minimize the communication overhead and to negotiate some parameters related to the data exchange that will occur. The CoSIP protocol is a constrained version of the SIP protocol intended to minimize the amount of network traffic, and therefore energy consumption, targeted for IoT scenarios. A similar effort in trying to minimize the amount of data in IoT and M2M applications is being carried on in standardization organizations, such as the IETF CoRE Working Group, which is currently defining a protocol (CoAP) to be used as a generic web protocol for RESTful constrained environments and maps to HTTP. Similarly, in this work we have proposed to apply the same approach to define a protocol for session instantiation, negotiation, and termination. We have described some interesting IoT scenarios that might benefit from using such a protocol, namely service discovery, session establishment, and services based on a subscribe/notify paradigm. A Java-language implementation of CoSIP has been developed and tested to evaluate the performance of the newly proposed protocol, by measuring the amount of transmitted bytes compared to other solutions based on SIP and CoAP respectively. The results show that applications that use CoSIP can outperform other SIP- and CoAP-based applications in terms of generated network traffic: SIP signaling can be compressed of nearly 50% using CoSIP, and long-running applications that may use CoAP for sending the same type of data to a given receiver may be better implemented with CoSIP, since no CoAP overhead has to be transmitted along with each transmitted data message leading to a packet size and per-packet processing reduction; packet size reduction in turn may reduce the need for packet fragmentation (in 6LoWPAN networks) and the energy consumption of the nodes involved in the data exchange.

Future work will include an exhaustive experimentation, both in simulation environments and a real-world testbed comprising a variety of heterogeneous devices which is currently being setup at the Department of

Information Engineering of the University of Parma, aimed to evaluate the performance of the CoSIP protocol both in terms of energy consumption and delay. The tests will focus on the time required to setup a session in different scenarios, such as in IEEE 802.15.4 multi-hop environments, and the measurement of energy consumption with a comparison between CoSIP sessions and standard CoAP communication. Two different perspectives will be analyzed: i) end-to-end delay between the actual session participants and ii) energy consumption on intermediate nodes which will be indirectly involved in the session as responsible for multi-hopping routing at lower layers. The target platforms will be both constrained and non-constrained devices for session participants and relay nodes, in order to provide a thorough evaluation comprising heterogeneous devices operating under different conditions.

REFERENCES

[1] C. Bormann, *Guidance for Light-Weight Implementations of the Internet Protocol Suite*, IETF Internet-Draft *draft-ietf-lwig-guidance* (February 2013), `http://tools.ietf.org/id/draft-ietf-lwig-guidance`

[2] S. Deering, and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, Internet Engineering Task Force, RFC 2460 (December 1998).

[3] IETF IPv6 over Low Power WPAN Working Group. `http://tools.ietf.org/wg/6lowpan/`

[4] IETF Routing Over Low power and Lossy networks Working Group. `http://tools.ietf.org/wg/roll/`

[5] IETF Constrained RESTful Environments Working Group. `http://tools.ietf.org/wg/core/`

[6] Z. Shelby, K. Hartke, K., and C. Bormann, *Constrained Application Protocol (CoAP)*, IETF Internet-Draft *draft-ietf-core-coap* (May 2013), `http://tools.ietf.org/id/draft-ietf-core-coap`

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, Internet Engineering Task Force, RFC 2616 (June 1999).

[8] J. Postel, *User Datagram Protocol*, Internet Engineering Task Force, RFC 768 (August 1980).

[9] K. Hartke, *Observing Resources in CoAP*, IETF Internet-Draft *draft-ietf-core-observe* (February 2013), `http://tools.ietf.org/id/draft-ietf-core-observe`

[10] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, *SIP: Session Initiation Protocol*, Internet Engineering Task Force, RFC 3261 (June 2002).

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, Internet Engineering Task Force, RFC 3550 (July 2003).

[12] V. Jacobson, and C. Perkins, *SDP: Session Description Protocol*, Internet Engineering Task Force, RFC 4566 (July 2006).

[13] A. B. Roach, *Session Initiation Protocol (SIP)-Specific Event Notification*, Internet Engineering Task Force, RFC 3265 (June 2002).

[14] Z. Shelby, S. Krco, and C. Bormann, *CoRE Resource Directory*, IETF Internet-Draft *draft-ietf-core-resource-directory* (June 2013), `http://tools.ietf.org/id/draft-ietf-core-resource-directory`

[15] A. Rahman and E. Dijk, *Group Communication for CoAP*, IETF Internet-Draft *draft-ietf-core-groupcomm* (July 2013), `http://tools.ietf.org/id/draft-ietf-core-groupcomm`

[16] Z. Shelby, *Constrained RESTful Environments (CoRE) Link Format*, Internet Engineering Task Force, RFC 6690 (August 2012).

[17] mjSIP project. `http://mjsip.org/`

[18] mjCoAP project. `http://mjcoap.org/`

[19] CoSIP project. `http://cosip.org/download/`

Edited by: Maria Fazio and Nik Bessis
Received: Nov 2, 2013
Accepted: Jan 10, 2014

DOI 10.12694/scpe.v14i4.932
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. 265–277. http://www.scpe.org

EVALUATING A FILE FRAGMENTATION SYSTEM FOR MULTI-PROVIDER CLOUD STORAGE

MASSIMO VILLARI, ANTONIO CELESTI, MARIA FAZIO, AND ANTONIO PULIAFITO *

Abstract. Currently, storage services represent a new way to do business in Cloud computing. This new trend is proved by the number of Cloud storage providers that are appearing on the market. In this work, we present an innovative approach useful for using different Cloud storage providers in a transparent way, avoiding both data lock-in and possible data privacy violation that can be caused by providers themselves. More specifically, we propose an approach enabling Cloud customers to rely on many Cloud storage providers. Differently from other solutions, with our approach only the customers have the full control of their data, and in addition, if a provider suddenly disappears and/or it is not available anymore, the customers will be able to continue accessing their data, reconstructing them from data fragments replicated in other Cloud storage providers. The paper shows how such an approach works. In particular, experiments, besides proving the goodness of our approach, also provide several guidelines regarding how to properly configure software systems in order to meet the customer's requirements (in terms of both QoS and costs).

Key words: Cloud Computing, Storage, Big Data, Reliability, Confidentiality.

1. Introduction. Nowadays, Cloud storage service is a very challenging topic, since it allows to store huge amount of data into different providers. The business behind the Cloud storage service is evident due the increasingly availability of storage providers (e.g., Dropbox, Google Drive, Copy, Amazon S3, SkyDrive, and so on). However, from the customer point of view, it is hard to choose the best offers, manage several accounts, and move data across multiple Cloud providers. Moreover, despite Cloud providers warranties, users' privacy could be compromised. From our point of view Cloud storage solutions lacks of a strong level of security and privacy [2], [3], [13]. In order to address such a problem, we propose to disseminate pieces of data among several Cloud providers that only the utilizer will be able to reconstruct.

In this work, we introduce an abstraction layer that works above heterogeneous Cloud storage providers. The benefits of the proposed strategies are multiple. Firstly, customers do not need to take care about a specific provider for data upload/download. They experience the storage service as a seamless service, where storage space is almost the sum of the storage spaces offered by the involved Cloud providers. Secondly, Cloud providers cannot have full access to the stored files, because each one is split in many chunks, that are stored into different Cloud providers. The technique we adopt in our approach that aims to avoid misusing of personal data is called *Data Obfuscation*.

Our solution is able to recover an rebuild the original file even if an error in a Cloud storage provider occurs (e.g., the operator fails). This is possible by means of the Redundant Residue Number System (RRNS) algorithm, that allows to split each file in several chunks that are called "residue-segments", including a redundancy code. Redundancy guarantees the recover of an original file when one or more residue-segments are missing. Before uploading data, the user selects the level of redundancy along with the Cloud providers involved in the storage service. Each residue-segment is BASE-64 encoded, attached within an XML wrapper and described through an XML metadata file, called *map-file*. Data and metadata are spread over different Cloud storage providers. The map-file tracks where the residue-segments are stored, in order to reconstruct the original file when the user requires to retrieve it. Following this approach, only the user can reconstruct the XML metadata combining the partial metadata coming from the two trusted providers, hence retrieve residue-segments, and rebuild the original files. This paper, extends our previous work [14], providing a more in-depth analisys about the proposed approach and further experimental results.

The paper is organized as follows. Section 2 describes related works, highlighting the lack of a resilient and confidential multi-provider Cloud storage service. Section 3 motivates this work according to the current trend on Cloud storage services. Section 4 briefly describe the RRNS algorithm on which our approach is based. Our solution for a reliable and confidential multi-provider Cloud storage service is described in Sect. 5. Experimental evaluations are discussed in Sect. 6. Finally, our conclusion are summarized in Sect. 7.

*DICIEMA, University of Messina,C.Da Di Dio 1, 98166, Messina, Italy (mvillari, acelesti, mfazio, apuliafito)@unime.it.

2. Related Work. Many works in literature deal with data reliability in data centers and in Cloud Infrastructure as a Service (IaaS). A well known solution is the Google File System (GFS), in which a file chunk replication mechanism is used [6]. Specifically, Google thought to make up a redundant storage of massive amounts of data on cheap and unreliable computers. The file chunk replication strategy is also at the basis of our solution.

In [1], the authors claim the improvement of file reliability by introducing redundancies into a large storage system exploiting different solutions, such as erasure correcting codes (used in RAID levels 5 and 6), introducing several data placement, failure detection and recovery disciplines inside data centers.

In [10], a data restore is accomplished using regenerating codes. In such a work, both redundancy and check controls are used to guarantee the possibility to repair data during file transfer over unreliable networks.

How to store pieces of file into Virtual Machines (VMs) is discussed in [11]. The authors introduce an enhanced distributed Cloud storage system. Nevertheless, the adopted protocol is rather complex and hard to be adapted in real scenarios. A similar technique is discussed in [15], where the authors present PRESIDIO, a framework able to detect similarity and reduce or eliminate redundancy when storing objects. The work in [7] discusses a way for optimizing the file partition in network storage environment. The model assessed by the authors shows how a partitioned network is able to maintain high availability. However, the approach is theoretical. In [9], a secure Cloud backup system is investigated. The authors study how to manage the Data Deletion (Assured) and the Version Control.

In [7], the authors describe a technique for optimizing the file partition considering a Network Storage Environment. They present a strategy to efficiently distribute files inside a cluster taking into account concepts of reliability, availability and serviceability. A file partitioning approach for Cloud computing is described in [4], where a smart procedure is used to optimize the placement of each data block according to its size. In [16], the authors face the problems that arise whenever a laptop is lost or stolen. The system guarantees that data cannot be accessed after an a priori configured time window and this can be a point of failure. The authors use XOR operations to split and merge files that have to be protected. The procedure is hard to be applied, because it requires to customize the kernel of the involved servers.

Data distribution [17] along with Data Migration [8] are topics quite relevant in the context of Cloud storage. The need to send big data over the Internet is important as well as the possibility to overcome data lock-in issues. Cloud operators are trying to prevent them for maintaining their business.

3. Motivations: the Current Trend on Cloud Storage. The arising requirement of Internet and, in particular, of Cloud Computing is the management of "Big Data". Big Data refers to a huge amount of data that users produce due the massive interconnection of smart devices and sensors over the Internet, which represent the basis for the development of Internet of Things (IoT) applications. Cloud Storage services represent the basic infrastructure for storing this produced data.

Cloud services are organized into three main levels: Infrastructure, Platform, or Software as a Service (i.e., IaaS, PaaS, or SaaS). Cloud providers can rely on these three levels in order to provide several storage functionalities. Amazon is the largest Cloud storage player and provides Storage as IaaS. Looking at the Amazon S3 storage service, it provides a simple web service able to store/retrieve any kind of data into/from the web. S3 provides the access to the scalable, reliable, secure, and fast Amazon storage infrastructure. S3 is widely used by many SaaS providers (e.g., Dropbox, Megauploader, Rapidshare, etc). In this paragraph, we discuss the trend of the number of objects stored in the S3 infrastructure during the last seven years.

The number of objects stored into the Amazon S3 has grown from roughly 700 Billion objects in one year to 2000 Billion objects at the time of when this paper is written. Figure 3.1 shows the market on data production and storage is vertiginously growing up in the last decade, according to the following equation:

$$NumObjs = 1,1335 * X^{3,5905} \tag{3.1}$$

Equation 3.1 allows to make the prevision on the storage demand of about 12000 Billion objects in the next five years (2018). Such a prevision is justified by the increasing interest of Information and Communication Technology (ICT) societies and end-users towards Cloud storage, with the purpose to reduce costs and satisfy their needs with a large plethora of opportunities. According to these considerations, we propose a new way to support emerging requirements for future Cloud storage.

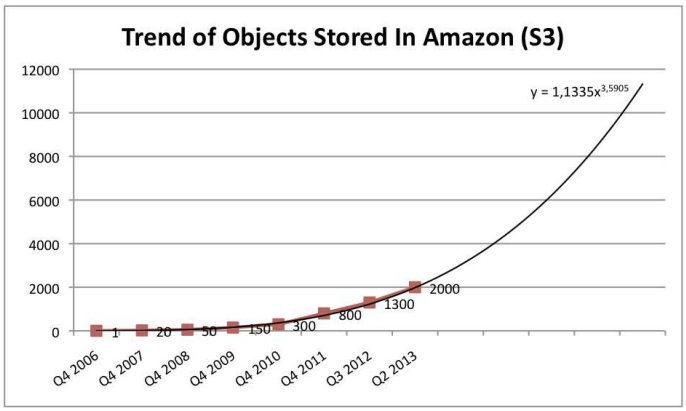
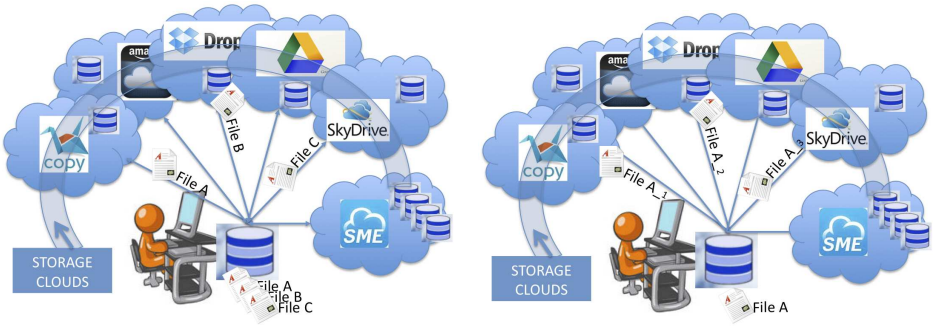


FIG. 3.1. *Amazon S3 growth in the last seven years and its future prevision for the next five years.*



(a) Classic management of data over multiple Cloud storage providers

(b) The new approach introduced in this work

FIG. 3.2. *Storage Cloud Services distributed over the Internet*

3.1. A New Approach to Store Data Into the Cloud. The usage of Cloud storage providers is characterized by the possibility for customers to subscribe many storage services even for free (e.g., Copy, DropBox, Drive,...) and to manually manage data upload and download. For example, a user holds files A, B and C in his local file system and decides to upload these files into the Cloud, as shown in Fig. 3.2(a). He/she has to choose where each file has to be uploaded (for example, file A in Copy, file B in Dropbox, and file C in SkyDrive), by using personal policies (such as type of file, size of file, and so on) or making a random choice. Thus, when he/she will require the file, he/she has to remember where the file has been uploaded. In addition, after upload, Cloud providers have the full control of the files they store, and this can be a potential threat for data confidentiality. We believe that a way for reducing such threats consists in using the Cloud storage in a different way than usual. In this paper, we describe an approach that consists in spreading chunks of each single file over different Cloud storage providers. Our approach introduces a software layer that abstracts heterogeneous Cloud storage providers and allows end-users to upload their files in an efficient way. Figure 3.2(b) shows how the proposed approach works. The original file A is split in three chunks, A_1, A_2 and A_3. The end-user makes a choice about the level of redundancy of each file, in order to overcome fails in data retrieval or data loss. The algorithm we implemented in our solution allows to manage a particular file redundancy in a smart way, as we will describe in Sect. 4.1. Of course, file splitting and merging processes are hidden to each Cloud operator, so that they cannot have any knowledge about data content. This prevents the possible fraudulent access to customer data by Cloud storage providers, that is a very hot topic on Cloud computing.

Moreover, pieces of file or chunks are wrapped into XML structure, in order to increase the portability of the system. Any Cloud storage provider sees the XML file with a body containing a chunk of the original file coded in BASE-64. In the example in Fig. 3.2(b), some chunks can be stored in Copy, others in SkyDrive, till Dropobox. Only end-users should be totally aware of what data are stored in each Cloud storage operator. Chunks distribution over the Cloud is described in a metadata map-file, an XML file that tracks where chunks are stored and allows to reconstruct the original file. The failure of the metadata map-file determines the loss of the whole file. To prevent this event and improve the reliability of the proposed solution, the map-file has to be stored in the Cloud, but information on chunk distribution must be spread over two or more further partial metadata map-files and deployed over two or more different independent trusted Cloud providers in order to carry out also medadata obfuscation. Since the trusted providers hold only partial metadata map-file, no one must be able, by itself, to reconstruct the whole metadata map-file of any particular user. In the following, we describe the mathematical concepts behind the redundancy algorithm used in our approach.

4. The Redundant Residue Number System. If you consider p prime, pairwise and positive integers $m_1, m_2, \cdots, m_p$ called *modulus* such as $M = \prod_{i=1}^{p} m_i$ and $m_i > m_{i-1}$ for each $i \in [2, p]$. Given $W \geq 0$, we can define $w_i = W \mod m_i$ the residue of W modulo m_i. The p-tuple $(w_1, w_2, \cdots, w_p)$ is named the *Residue Representation* of W with the given modulus and each tuple element w_i is known as the ith residue digit of the representation. For every p-tuple $(w_1, w_2, \cdots, w_p)$, the corresponding W can be reconstructed by means of the Chinese Remainder Theorem:

$$W = \left(\sum_{i=1}^{p} w_i \frac{M}{m_i} b_i \right) \mod M \tag{4.1}$$

where b_i, $i \in [1, p]$ is such that $\left(b_i \frac{M}{m_i} \right) \mod m_i = 1$ (i.e. the multiplicative inverse of $\frac{M}{m_i}$ modulo m_i). We call *Residue Number System (RNS)*, with residue modulus $m_1, m_2, \cdots, m_p$, the number system representing integers in $[0, M)$ through the p-tuple $(w_1, w_2, \cdots, w_p)$. Considering $p + r$ modulus $m_1, \cdots, m_p, m_{p+1}, \cdots, m_{p+r}$ we have:

$$M = \prod_{i=1}^{p} m_i \tag{4.2}$$

and

$$M_R = \prod_{i=p+1}^{r} m_i \tag{4.3}$$

without loss of generality $m_i > m_{i-1}$ for each $i \in [2, p + r]$. We define *Redundant Residue Number System (RRNS)* of modulus $m_1, \cdots, m_{p+r}$, range M and redundancy M_R, the number system representing integers in $[0, M)$ by means of the $(p + r)$-tuple of their residue modulus $m_1, \cdots, m_{p+r}$. Although the above mentioned *RRNS* can provide representations to all integers in the range $[0, M \cdot M_R)$, the legitimate range of representation is limited to $[0, M)$, and the corresponding $(p+r)$-tuples are called *legitimate*. Integers in $[M, M \cdot M_R)$ together with the corresponding $(p+r)$-tuples are instead called *illegitimate*. Let now consider an *RRNS* whose range is M and redundancy M_R, where $(m_1, m_2, \cdots, m_p, m_{p+1}, \cdots, m_{p+r})$ is the $(p+r)$-tuple of modulus and $(w_1, w_2, \cdots, w_p, w_{p+1}, \cdots, w_{p+r})$ is the legitimate representation on an W integer in $[0, M)$. If an event maks unavailable d arbitrary digits in the representation, we have two new sets of elements $\{w'_1, w'_2, \cdots, w'_{p+r-d}\} \subseteq \{w_1, \cdots, w_{p+r}\}$ with the corresponding modulus $\{m'_1, m'_2, \cdots, m'_{p+r-d}\} \subseteq \{m_1, \cdots, m_{p+r}\}$. This status is also known as *erasures* of multiplicity d. If the condition $d \leq r$ in true, the *RNS* of modulus $\{m'_1, m'_2, \cdots, m'_{p+r-d}\}$ has range:

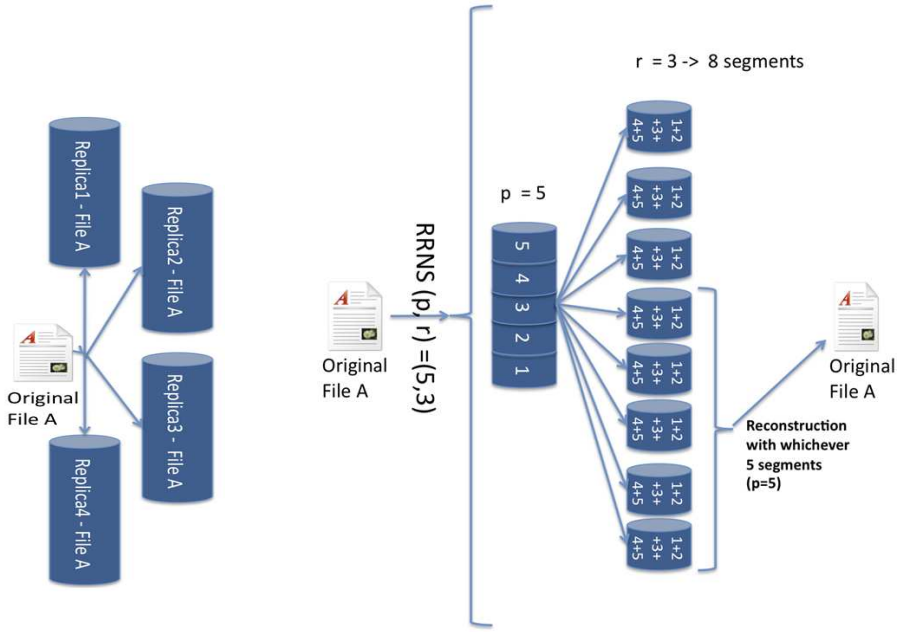
$$M' = \prod_{i=1}^{p+r-d} m'_i \leq M \tag{4.4}$$

since $W < M$, $(w_1, w_2, \cdots, w_p, w_{p+1}, \cdots, w_{p+r})$ is the unique representation of W in the latter *RNS*. Integer W can be reconstructed from the $p+r-d$-tuple $(w'_1, w'_2, \cdots, w'_p, w'_{p+1}, \cdots, w'_{p+r-d})$ by means of the Chinese Remainder Theorem (as in the case of equation 4.1):

$$W = \left(\sum_{i=1}^{p+r-d} w'_i \frac{M'}{m'_i} b'_i \right) \mod M' \tag{4.5}$$

where b_i is such that $\left(b'_i \frac{M'}{m'_i} \right) \mod m'_i = 1$ and $i \in [1, p+r-d]$. As a consequence, the above mentioned *RRNS* can tolerate erasures up to multiplicity r. It can be proved (see [12] for further details) that the same *RRNS* is able to detect any error up the multiplicity r and it allows to correct any error up the multiplicity $\lfloor \frac{r}{2} \rfloor$.

In the following, we are going to explain the overhead introduced by the RRNS encoding algorithm.



(a) Classic management of data redundancy.

(b) File management based on RRNS

FIG. 4.1. *Traditional approach versus the RRNS-based approach.*

4.1. RRNS Exploitation for Cloud storage. In this paper we present a new solution for storing files into the Cloud based on RRNS. It works by using encoding techniques and redundancy policies to offer a very reliable and more secure service using obfuscation. Of course, from the point of view of the storage overhead, the RRNS cause an increase in terms of file size. Here, we analyze the impact of the proposed solution in terms of space disk overhead, comparing our results with more traditional approaches.

The traditional approach used to increase fault tolerance in data storage is to replicate data, as shown in Fig. 4.1(a). Thus, if we need a 3 degree of redundancy for file A (that means we can recover A even if 3 files are lost), we need to deploy 4 replicas of A in different Cloud storage providers. The redundancy mechanism implemented in RRNS allows to reduce the storage consumption. Let consider the following two parameters: p

is the minimum number of modules necessary to reconstruct a file and r is the redundancy degree. Let consider a generic file A. We split A in p residue-segments. Moreover, to have 3 degree of redundancy, we set $r = 3$ and we can recover A even if 3 residue-segment are lost (Figure 4.1(b) shows the behavior of the system for p=5). According to formulations expressed in Sect. 4, a system able to tolerate chunks unavailability up to $d = 3$ can be codified with different configurations:

- p=1, r=3 → that is 4 residue-segments;
- p=3, r=3 → that is 6 residue-segments;
- p=5, r=3 → that is 8 residue-segments;
- p=7, r=3 → that is 10 residue-segments;
- p=9, r=3 → that is 12 residue-segments.

The difference among the above configurations depends on the number of Cloud storage operators available to store replicas/segments. By using RRNS, we can modulate the number of devices (i.e., Cloud storage providers) that have to be involved for storing data, that is 4 (as the traditional case), 6, 8, 10 and 12. This allows us to increase the overall reliability of the system if we consider many Cloud operators. In Fig. 4.1(b), the original file is split into 8 residue-segments. Here, the erasure r is equal to 3. The system is able to reconstruct the original file up to $d = 3$ residue-segments unavailability.

At the end of the encoding/uploading process, a single Cloud storage provider holding the whole file will not exist and this will lead to some direct consequences: even though there's not encryption on data, a self-contained file will not exist on any storage provider, leading to an increased confidentiality degree. This type of data access restriction is also know as *data obfuscation*. Thanks to the redundancy introduced by the RRNS, in case of temporary unavailability of one or more residue-segments (according to Eq. 4.5), the user's file might still be reconstructed from the owner. Indeed, only the owner knows the logical distribution of segments stored over the different Cloud storage providers, hence their potential reconstruction. Obviously, the introduced redundancy increases the resulting amount of data to be stored and transferred, but how previously discussed such an overhead is acceptable compared with the traditional approach consisting on managing whole replicas of files. In addition, if a particular Cloud storage provider is heavily overwhelmed from users' requests, having data spread over different Cloud operators might quicken the download task. In particular, instead of waiting for the transmission of a monolithic block from the overloaded Cloud provider, the client can download different residue-segments in parallel from different operators, allowing a more efficient bandwidth occupation. This same method is used by the *Torrent* protocol for increasing the speed of file downloading over the Internet. In the following, we formalize the overhead of our approach with respect to the traditional one.

4.2. Overhead Evaluation for Reliable Capabilities. Let us consider a file management based on RRNS(p,r) and x as the original file size of file A. The base64 encoding used to make the XML wrapping implies that a set of 6 bits is converted in an ASCII character (8 bit), with a total overhead of $\frac{8-6}{6} = \frac{1}{3}$ [5]. Considering that 4 is the compression rate due to the RRNS algorithm, the final size S_{file} of the files we have to upload over the Cloud can be calculated as following:

$$S_{file} = (\frac{p+r}{4}) * x + (\frac{p+r}{4}) * \frac{1}{3} * x \tag{4.6}$$

where the first term is related to the increased size of the file due to the RRNS algorithm and the second term is the increased size of the overhead for the base64 encoding due to the RRNS algorithm. From equation 4.6 we obtain:

$$S_{file} = (\frac{p+r}{4}) * 1,33 * x \tag{4.7}$$

Thus, the storage size of the file depends on both p and r. In Fig. 4.2, we have drawn the multiplicative factor of the file size for a file of 1MB, when $p = 5$. For example, when $r = 7$, the storage size required is about 4MB. A traditional redundancy approach, where multiple copies of the file are stored, implies a storage size of 7MB. Thus, on equal error tolerance, our approach reduces the storage size of about a factor $1,75$.

5. RRNS-Based Approch for a Reliable Cloud Storage. We underline that the two key-points on which our approach is based consist in guaranteeing data availability (resiliency) and increasing data confidentiality through the *obfuscation technique*. From the client point of view, different types of application front-end

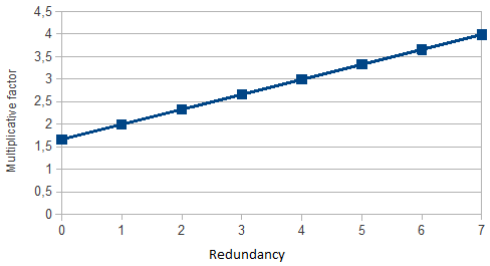


FIG. 4.2. *Multiplicative factor of file size with the redundancy*

can be developed: web application in form of SaaS, java applets, stand-alone desktop application, mobile application, and so on. Despite of any particular type of front-end application, here, we describe the main steps required to split a file and upload residue-segments over different Cloud operators and the main steps to recover residue-segments, downloading them, and reconstruct the original file. An application front-end should be able to receive as input one or more files belonging to a given user and uploading them over the several Cloud storage providers according to specific constraints. Thanks to the RRNS properties discussed in Sect. 4, each time the encoding process is applied to a file, it is split (as depicted in Fig. 5.1) on different residue-segments according to a given degree of redundancy. In order to guarantee a high level of flexibility, we use a XML wrapper for representing residue-segments adopting the BASE-64 encoding. BASE-64 encoding schemes are commonly used when there is the need to encode binary data that needs to be stored and transferred over media that is designed to deal with textual data. Although the XML container allows a strong level of environment independence, the BASE-64 encoding employed for encapsulating binary data within the *content node* involves a storage size overhead. In fact, the user data after the BASE-64 encoding will involve a storage requirement increment approximately of 33% as described in Sect. 4.2. Then, the XML residue-segments will be copied and stored on different Cloud storage providers by using the APIs they make available for developers.

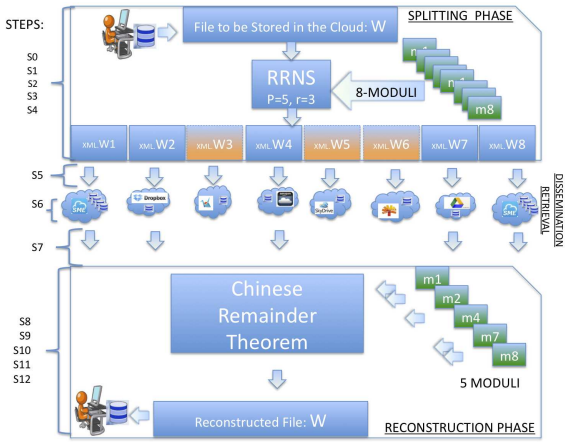


FIG. 5.1. *Representation of the RRNS encoding/decoding.*

After having introduced the general concepts regarding our idea, in the following we are going to discuss the a few implementation guidelines, analyzing how data is processed both at upload and download phases. Figure 5.1 depicts how these tasks are carried out: the top part represents the RRNS encoding/uploading process, while the underside one represents the downloading/decoding process. The main steps that have to be executed by a generic front-end application are schematized in Table 5.1. Whenever an end-user wants to store a file W, he/she specifies his/her requirements through an *init* file, in which he/she sets the parameters p and r and provides information on Cloud storage providers involved in data uploading (STEP S0). The

SPLITTING PHASE
S0: end-user selects the File W, the number of required fragments (p) and the redundancy degree (r). **S1:** $W_{zip} = ZIP(W)$. The the File W is compressed by means of the zip algorithm **S2:** $Wx = RRNS(W_{zip})$. This process generates $p + r$ residue-segments. **S3:** Wx BASE-64 encoding. **S4:** Wx XML encapsulation.
DISSEMINATION PHASE
S5: Upload of XML residue-segments. This step involves several Cloud storage operators. According to the particular type of front-end application, the upload task can be accomplished after S0. **S6:** Storage of XML residue-segments into several Cloud storage providers. Each provider is not able to know the content of the whole file.
RETRIEVAL PHASE
S7: Download of the XML chunks.
RECONSTRUCTION PHASE
S8: Wx XML decapsulation. **S9:** Wx BASE-64 decoding. **S10:** $W_{zip} = RRNS(Wx)$. The Zipped file is reconstructed using the Chinese Reminder Theorem. **S11:** $W = UNZIP(W_{zip})$. This compressed file is uncompressed with ZIP utility. **S12:** end-user access the original file. According to the particular type of front-end application, S8, S9, and S10 steps can be accomplished after S8.

TABLE 5.1
Main steps that have to be accomplished by a front-end application.

application compress the W file with the ZIP utility in order to save space (STEP S1). The RRNS encoding is applied to the zipped file and generate a set of $p + r$ residue-segments (STEP S2). Each residue-segment is BASE-64 encoded (STEP S3) and attached within an XML wrapper (STEP S4). Each XML wrapper, in turn, will then be uploaded to a particular Cloud storage providers specified by the user (STEP S5). Retrieval and reconstruction of the file W are performed thorough vice versa activities (STEPS S6-S12).

5.1. XML Wrapper Details. In order to track the location of uploaded residue-segments, for each file, a metadata map-file is created. Although, how previously stated, how to obfuscate the metadata map-file spreading it over different Cloud providers is out of scope, here we provide a few details about the structure of such a file also presenting a simple example of metadata map-file obfuscation using the MD5 and two different trusted Cloud providers. The metadata map-file must be accessible only from the data owner and it allows to rebuild the original file. In the following is presented an example of possible metadata map-file:

```
<OWNER>ownerInfo</OWNER>
<SEGMENTS>...</SEGMENTS>
<FILE>
    [...]
    <CHUNK num="11">Path/to/the/StorageProviderX/
            94090e1381a1700fb8c34a0069bc6533.xml</CHUNK>
    <CHUNK num="5">Path/to/the/StorageProviderY/
            eaf2bcdcb47cd1eba2a4392857e66b33.xml</CHUNK>
    [...]
</FILE>
```

The first element of the file, *OWNER*, specifies owner information. The *SEGMENTS* element includes the number of necessary segments to reconstruct the file (that is the value of p). The *FILE* element contains a variable

number of *CHUNK* elements. The *CHUNK* tag has the attribute *num*, which refers to the residue-segment sequence number, its content represents a combination of the path on associated on the front-end application, the Cloud storage provider for that chunk, and the name of the XML file containing data. Information stored within the above XML document will allow to build up the original file during the decoding process. Depending on the number of available providers and the number of XML chunks, providers are in charge to store one or more chunks.

It is straightforward foreseeing that the metadata map-file represents a key point of the whole process: its accidental lost or unavailability leads to data loss, because retrieving chunks and rebuilding the file becomes impossible. Thus, keeping the map-file in the local file system of the end-user is not a strategic solution. To improve the reliability of the metadata map-file storing, the front-end application has to be designed to also store the map-file into several Cloud providers. To preserve data confidentiality, the map-file has to be slit in different partial medatada map-files and spread over different independent trused Cloud providers. This mechanism can be achieved using well known security techniques, in particular combining asymmetric and/or symmetric encryption with the MD5 message-digest algorithm. In order to clarify ideas, in the following we discuss a methodology to split the metadata map-file into two partial metadata map-files using the MD5. In this example, partial metadata map-files are called *servicelist* and *trusted*. The *servicelist* file is an XML document containing the list of storage providers on which a user holds an account for uploading/downloading files. The *trusted* file in an XML document used to associate a residue-segment number to the name of the related XML file containing the residue-segment data, and an unique identifier associated to the service provider on which that chunk is stored:

```
[...]
<OWNER>ownerInfo</OWNER>
<SEGMENTS>...</SEGMENTS>
<FILE>
[...]
<CHUNK num="11">
    <CHUNK_REF>94090e1381a1700fb8c34a0069bc6533.xml</CHUNK_REF>
    <UUID_REF>a72ebba5d9b695c39e6d2193c3cb8057</UUID_REF>
</CHUNK>
<CHUNK num="5">
    <CHUNK_REF>eaf2bcdcb47cd1eba2a4392857e66b33.xml</CHUNK_REF>
    <UUID_REF>9e296f8ea3992ef53ff93e9adbc80299</UUID_REF>
</CHUNK>
[...]
</FILE>
[...]
```

The first two elements of the file are identical to the ones in the map-file. The *CHUNK* tag within the *FILE* element has the attribute *num*, which refers to the fragment order and the child elements *CHUNK_REF* and *UUID_REF* respectively identify the name of the XML file containing the data associated to that chunk and a unique identifier associated to the storage service provider. The *UUID_REF* does not contain the actual service provider in order to obfuscate this information to the storage provider where the file will be uploaded. In fact, the unique identifier is obtained applying the MD5 to the couple (chunk , provider) in order to prevent brute force attacks aiming at found out the actual paths associated to the chunks. Uploading to different trusted Cloud providers the two partial metadata files instead of the actual whole metadata map-file guarantees only to the end-user the knowledge about the file partitioning. This task is highlighted in Fig. 5.2. Starting from *serviceList* and *trusted* files, we can obtain the metadata map-file necessary for reconstructing the original file. The metadata map-file obfuscation can be obtained also with other techniques considering several trused Cloud providers. Further details about metadata map-file obfuscation are out of scope. In the following, we specifically focus on evaluating how the RRNS algorithm works considering multiple Cloud storage providers.

6. Experimental Results. In order to evaluate our system, we conducted several experiments considering a real testbed composed of client stand-alone java desktop front-end application interacting with three different commercial Cloud storage providers, that are Google Drive, Dropbox and Copy. In our experiments, we used file with different sizes, different redundancy factors and we evaluated the time spent for the splitting activity,

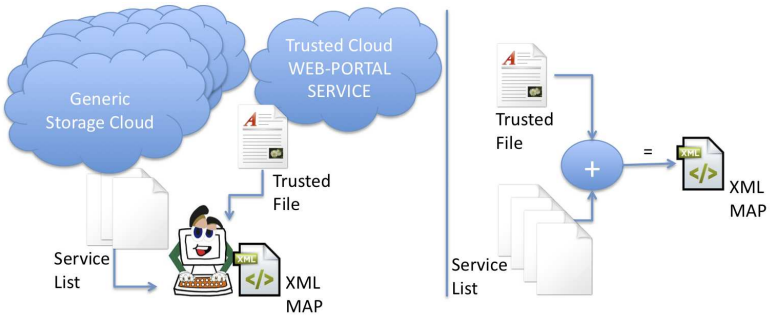
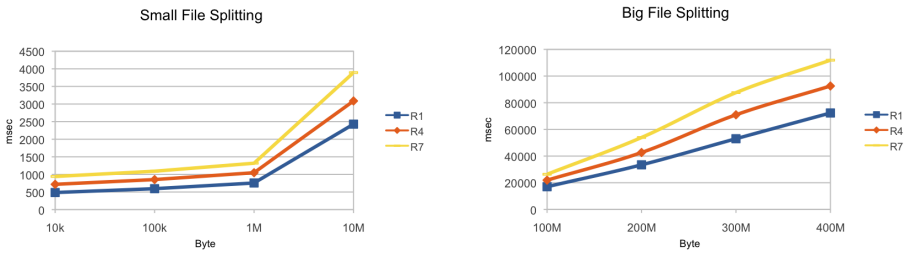


FIG. 5.2. *map file reconstruction.*

the time spent for uploading residue-segments, and the time spent for retrieving and re-composing the original file.

6.1. Testbed Setup. The local testbed was arranged at DICIEAMA GRID Laboratory at the University of Messina. The front-end application was deployed in a blade with the following hardware configuration: CPU Dual-Core AMD Opteron(tm) Processor 2218 HE, RAM 6GB, OS: ubuntu server 12.04.2 LTS 64 BIT. We considered two sets of file sizes defined *Small Files*, characterized by 10KB, 100KB, 1MB, 10MB, and *Big Files*, characterized by 100MB, 200MB, 300MB, and 400MB. We fix p=5 and the following values for r: [1, 4, 7]. According to Eq. 4.5, each file is split respectively into 6, 9, and 12 residue-segments, from now on called chunks. We store chunks balancing the workload over the three Cloud storage providers, so that in each one, we stored respectively 2, 3, and 4 pieces of file. Each experiment was repeated 30 times in order to consider mean values and confidence intervals at 95%. In the following, we will describe the performance of Splitting and Reconstruction phases along with Dissemination and Retrieval phases.



(a) Time needed for the Splitting phase with Small Files

(b) Time needed for the Splitting phase with Big Files

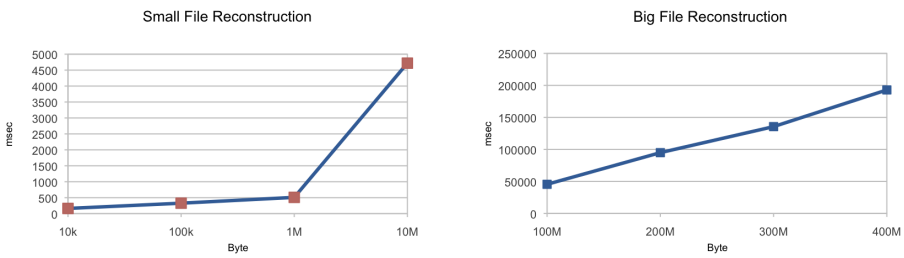
FIG. 6.1. *Time needed for the Splitting phase with Small and Big Files.*

6.2. Performance Analysis for Splitting and Reconstruction Phases. Here, we investigate the behavior of Splitting and Reconstruction phases described in Table 5.1. They are quite similar, but the Reconstruction phase takes into account a reduced number of chunks, i.e., $p = 5$, that are downloaded and reconstructed.

Figure 6.1 summarizes the time required to split each file. The x-axis reports the file size in bytes, whereas the y-axis reports the time in milli-seconds (msec). The graphs show the trend for both Small and Big files. For file sizes up to 1MB, the processing time is almost constant considering the different file sizes. Increasing the file size from 10 MB to 400MB, the time increasing is quite double at each step. Different values for r do not

significantly affect the performance, hence the user can improve the reliability of the storage service without a high degradation of performance.

Similar results, depicted Fig. 6.2, show the time spent for rebuilding Small and Big files. The time required to reconstruct a big file size, i.e, 400MB is about 200 seconds. The reconstruction phase is much more heavy respect to the split phase (110 seconds for a 400MB file with $r = 7$). Different values of r are not considered because the minimum number that is used for file reconstruction is $p = 5$.



(a) Time needed for the Reconstruction phase with Small Files

(b) Time needed for the Reconstruction phase with Big Files

FIG. 6.2. *Time needed for the Reconstruction phase with Small and Big Files.*

6.3. Performance Analysis for Dissemination and Retrieval Phases. Here, we investigate the behavior of Dissemination and Retrieval phases described in Table 5.1. Figure 6.3 shows the time spent to send

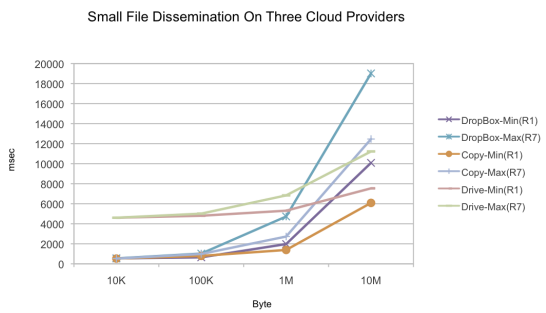


FIG. 6.3. *Time spent for the dissemination process to three Cloud providers.*

chunks in parallel to Google Drive, Copy, and Dropbox. We analyzed the transfer time of files up to sizes of 10MB with $r = 1$ and $r = 7$ respectively, i.e., the minimum and maxim values for r. The y-axis reports the time in milliseconds (msec), whereas the x-axis reports the file size in bytes. In these proofs, we prefer to limit the file size to 10 MB because in our testbed the upload bandwidth was much more low respect to the download one. This allowed us to repeat more times experiments in a reasonable time interval. For files of 10KB and 100KB, we experienced very similar transfer times and Google Drive results the slowest. For files of 10KB with r=1 the transfer times in Copy and Dropbox take respectively in average 523 msec and 543 msec. With Google Drive, the transfer time instead takes in average 4603 msec. We observed a similar trend considering r=4 and r=7 as well.

As the file size grows up over 1MB, results changed. Google Drive is the slowest provider again, but the transfer times increases considering different redundancy factors. Analyzing the results, we distinguished different behaviors between Copy and Dropbox: the former resulted the most efficient, instead the latter started

to degrade in performance. For files of 10MB, we observe an interesting behavior: Google Drive becomes more efficient than Copy and Dropbox, that experienced performance degradation. In fact, with r=7 the transfer times took 11223 msec, 12461 msec, and 19015 msec respectively with Google Drive, Copy and Dropbox. This means that for small file sizes ($< 100KB$) Copy and Dropbox are more efficient than Google Drive, but for big file ($> 10MB$) Drive is more efficient than Copy and Dropbox. In particular, Copy has a trend slightly worse than Google Drive, instead Dropbox results absolutely the worst. Figure 6.4 analyzes the download time of

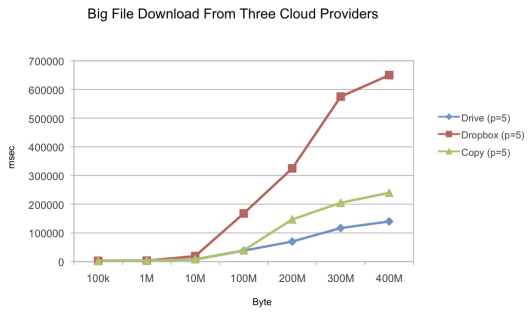


FIG. 6.4. *Time spent to download chunks from three different Cloud providers*

files with size from 100K up to 400 MB . For simplicity, we considered 5 segments ($p = 5$) stored on the same operator since performance evaluation is more complex mixing different operators. The picture highlights the better behavior of Google Drive respect to the others. In the worst case, the time for downloading 5 chunks for a file of 400 MB is more of 10 minutes (65000 msec). With the other providers, it is necessary to spend less than 5 minutes (respectively 140000 msec and 240000 msec). These results are reasonable enough for guarantying a good user experience in using the the our system.

This analysis allowed us to know the behavior of three of the major Cloud storage providers in order to understand a few useful information about how setup a system using different Cloud storage providers.

7. Conclusion and Future Works. In this paper, we discussed the data reliability and confidentiality problems considering a multi-provider Cloud storage service. By means of RRNS, our approach allows to split a file in p+r chunks, which are deployed over different Cloud storage operators. The advantage of our approach is twofold: on one hand, each single provider cannot access the whole file, and on the other hand if a provider is not available, files can be retrieved considering p pieces of files stored in other operators. Experiments highlighted how both file size and redundancy degree impact the performance of the proposed system considering Google Drive, Dropbox, and Copy. In future works, we aim to better investigate such an approach also considering data encryption.

REFERENCES

[1] Deepavali Bhagwat, Kristal Pollack, Darrell D. E. Long, Thomas Schwarz, Ethan L. Miller, and Jehan-Francois Paris. Providing high reliability in a minimum redundancy archival storage system. In *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, MASCOTS '06, pages 413–421, Washington, DC, USA, 2006. IEEE Computer Society.

[2] A. Celesti, M. Fazio, and M. Villari. Se clever: A secure message oriented middleware for cloud federation. In *IEEE Symposium on Computers and Communications (ISCC)*, 2013.

[3] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. How the dataweb can support cloud federation: Service representation and secure data exchange. In *Second Symposium on Network Cloud Computing and Applications (NCCA)*, pages 73–79, 2012.

[4] Kai Fan, Libin Zhao, Xuemin Shen, Hui Li, and Yintang Yang. Smart-blocking file storage method in cloud computing. In *2012 1st IEEE International Conference on Communications in China (ICCC)*, pages 57–62, 2012.

[5] N. Freed and N. Borenstein. MIME: Multipurpose Internet Mail Extensions. Technical Report RFC2045, 1996.

[6] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system , 2013. http://www.cs.umd.edu/class/spring2011/cmsc818k/Lectures/gfs-hdfs.pdf.

[7] Wu Hai-Jia, Liu Peng, and Chen Wei-wei. The optimization theory of file partition in network storage environment. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 30–33, 2010.

[8] P. Nahar, A. Joshi, and A. Saupp. Data migration using active cloud engine. In *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on*, pages 1–4, 2012.

[9] A. Rahumed, H.C.H. Chen, Yang Tang, P.P.C. Lee, and J.C.-S. Lui. A secure cloud backup system with assured deletion and version control. In *Parallel Processing Workshops (ICPPW), 2011 40th International Conference on*, pages 160–167, 2011.

[10] K.W. Shum and Yuchong Hu. Functional-repair-by-transfer regenerating codes. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 1192–1196, 2012.

[11] S. Srivastava, V. Gupta, R. Yadav, and K. Kant. Enhanced distributed storage on the cloud. In *Computer and Communication Technology (ICCCT), 2012 Third International Conference on*, pages 321–325, 2012.

[12] N. S. Szabo and R. I. Tanaka. *Residue Arithmetic and its Applications to Computer Technology.* Mc Graw-Hill, New York, 1967.

[13] G Vernik, A. Shulman-Peleg, S. Dippl, C. Formisano, M.C. Jaeger, E.K. Kolodner, and M. Villari. Data on-boarding in federated storage clouds. In *IEEE 6th International Conference on Cloud Computing*, 2013.

[14] Massimo Villari, Antonio Celesti, Francesco Tusa, and Antonio Puliafito. Data reliability in multi-provider cloud storage service with rrns. In Carlos Canal and Massimo Villari, editors, *Advances in Service-Oriented and Cloud Computing*, volume 393 of *Communications in Computer and Information Science*, pages 83–93. Springer Berlin Heidelberg, 2013.

[15] Lawrence L. You, Kristal T. Pollack, Darrell D. E. Long, and K. Gopinath. Presidio: A framework for efficient archival data storage. *Trans. Storage*, 7(2):6:1–6:60, July 2011.

[16] Nan Zhang, Jiwu Jing, and Peng Liu. Cloud shredder: Removing the laptop on-road data disclosure threat in the cloud computing era. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 1592–1599, 2011.

[17] Yu Zhang, Weidong Liu, and Jiaxing Song. A novel solution of distributed file storage for cloud service. In *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*, pages 26–31, 2012.

Edited by: Maria Fazio and Nik Bessis
Received: Nov 2, 2013
Accepted: Jan 10, 2014

DOI 10.12694/scpe.v14i4.933
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. 279–290. http://www.scpe.org

ANALYTICAL INVESTIGATION OF AVAILABILITY IN A VISION CLOUD STORAGE CLUSTER

DARIO BRUNEO[†], FRANCESCO LONGO[†], DAVID HADAS[‡], AND ELLIOT K. KOLODNER[‡]

Abstract. The goal of VISION Cloud, a European Commission funded project, is to design a new scalable and flexible storage cloud architecture able to provide data-intensive storage cloud services. The proposed environment employs a distributed file system on top of a set of storage rich nodes composing a cluster. Several clusters constitute a data center, while multiple geographically distributed data centers form a single storage cloud. In this paper, we focus on a single VISION Cloud storage cluster, providing a stochastic reward net model for an investigation of its availability. The proposed model is a first attempt at obtaining a quantification of the availability level of the cloud storage provided by the VISION Cloud architecture.

1. Introduction. Focusing on IT assets as commodities and on-demand usage patterns, cloud computing greatly mitigates the cost of service provisioning, through tools such as virtualization of hardware, rapid service provisioning, scalability, elasticity, accounting granularity, and cost allocation models. However, Future Internet, Internet of Things, and, in general, the rich digital environment we are experiencing nowadays pose new requirements and challenges in the Cloud area, especially with respect to the explosion of personal and organizational digital data. In fact, the strong proliferation of data-intensive services and the digital convergence of telecommunications, media, and ICT will surely amplify the explosion of raw data and the dependence on data services. System performance and dependability [3, 15], energy consumption [4], workload characterization [9] are only few examples of the Cloud related research trends that have been investigated in the last years.

VISION Cloud [11] is a European Commission Seventh Framework Programme (FP7/2006-2013) funded project. Its goal is to design a new scalable and flexible storage cloud architecture that allows the implementation of data-intensive storage cloud services, scalability and flexibility referring to the ability of the proposed architecture to deal with a large number of concurrent users and in allowing the provisioning of different kinds of storage services. Raising the abstraction level of storage, enabling data mobility across providers, allowing computational storage and content-centric access to storage and deploying new data-oriented mechanisms for QoS and security guarantees are some of the means that VISION Cloud exploits in order to achieve such a goal. With respect to QoS guarantees, reliability, availability, and fault tolerance and resiliency characteristics of the provided services are important aspects that need to be taken into consideration.

The single storage resource in the VISION Cloud reference architecture is represented by the *storage cluster* which usually includes hundreds of storage rich nodes. Such a basic element is able to store data objects and provide computational power on top of it in a transparent way. This is obtained by the use of a distributed file system installed on the storage cluster. In the prototype implementation of the architecture that the VISION Cloud project provides, the General Parallel File System for Shared Nothing Clusters* (GPFS-SNC) [8] is exploited. A high level of availability and resiliency to faults is achieved by replicating data objects across different storage clusters. VISION Cloud considers a single cloud as composed by multiple distributed data centers interconnected through dedicated networks. Each data center can be composed of multiple storage clusters.

In this paper, we provide an analytic model for the availability investigation of a storage cluster in the context of the storage cloud environment proposed by the VISION Cloud project. The model is based on stochastic reward nets (SRNs) [6], an extension of generalized stochastic Petri nets. SRNs are a graphical tool for the formal high-level representation of systems characterized by concurrency, mutual exclusion, conflict, and synchronization dynamics. Thus, such a formalism is useful in capturing the key concepts of large-scale distributed systems [5, 2] and the model we propose allows obtaining information about the reached availability level of a VISION Cloud storage cluster varying both structural and timing system parameters. Structural parameters are related to the number of nodes in the cluster, the number of disks in each node, the cluster file system metadata replication level, and similar information. Timing parameters involve information about the

[†]Dipartimento di Ingegneria DICIEAMA, Università degli Studi di Messina, Messina, Italy ({dbruneo,flongo}@unime.it)
[‡]IBM Research Labs Haifa, Haifa, Israel, ({kolodner,davidh}@il.ibm.com)
*GPFS is a trademark of International Business Machines Corp., registered in many jurisdictions worldwide.

time necessary to specific events (e.g., disk or node failure) to occur or specific operations (e.g., disk or node repair, cluster file system metadata recovery) to be performed.

Prior work deals with the performance analysis of storage cloud infrastructure [13] while little effort has been put in the context of availability analysis [20]. In this context, the majority of the work mainly considers replica placement policies [12, 19] without taking into consideration real case studies as done in our work. In fact, our model could be exploited by a VISION Cloud administrator in order to opportunely build the infrastructure accordingly to the desired availability level both from the hardware (e.g., computation, storage, network resources) and the software (e.g., replication schema, cluster file system configuration) points of view. Moreover, it could represent an useful instrument for model assisted SLA management.

The paper is organized as follows. Section 2 gives a background about Petri nets with particular reference to SRNs. Section 3 provides an overview of the VISION Cloud reference architecture and illustrates how GPFS-SNC is exploited in the reference implementation. Section 4 formally describes the considered scenario while Section 5 illustrates how such a scenario is modeled through the use of SRNs. Section 6 provides some numerical results. Finally, Section 7 concludes the paper with some final remarks on the proposed approach and on possible future work.

2. Background about Petri Nets. A Petri net (PN) [16] is a 4-tuple: $PN = (P, T, A, M)$, where P is the finite set of *places* (represented by circles), T is the finite set of *transitions* (represented by bars), A is the set of *arcs* (connecting elements of P and T) and M is the set of markings each of which denotes the number of tokens in the places of the net. Graphically, a PN is a directed bipartite graph, with two types of nodes: *places* and *transitions*. A directed arc connecting a place (transition) to a transition (place) is called an input (output) *arc* of the transition. A positive integer called multiplicity can be associated with each arc. Each place may contain zero or more tokens. A transition is *enabled* if each of its input places has at least as many tokens as the multiplicity of the corresponding input arc. A transition can *fire* when it is enabled, and upon firing, a number of tokens equal to the multiplicity of the input arcs is removed from each of the input places, and a number of tokens equal to the multiplicity of the output arcs is deposited in each of its output places. In stochastic Petri net (SPN), exponentially distributed firing times can be associated to the net transitions so that the stochastic process underlying a SPN is a homogeneous CTMC. In generalized stochastic Petri nets (GSPN) [14], transitions are allowed to be either *timed* (exponentially distributed firing time, drawn as rectangular boxes) or *immediate* (zero firing time, represented by thin black bars). Immediate transitions always have priority over timed transitions and if both timed and immediate transitions are enabled in a marking then timed transitions are treated as if they are not enabled. If several immediate transitions compete for firing, a specified probability mass function is used to break the tie. A marking of a GSPN is called *vanishing* if at least one immediate transition is enabled in it. A marking is called *tangible* otherwise. GSPN also introduces the concept of *inhibitor arc* (represented by a small hollow circle at the end of the arc) which connects a place to a transition. A transition with an inhibitor arc can not fire if the input place of the inhibitor arc contains more tokens than the multiplicity of the arc. SRNs [6] are extensions of GSPNs. In SRNs, every tangible marking of the net can be associated with a reward rate thus facilitating the computation of a variety of performance measures. Key differences with respect to GSPNs are: (1) each transition may have an enabling function (also called a guard) so that a transition is enabled only if its marking-dependent enabling function is true; (2) marking dependent arc multiplicities are allowed; (3) marking dependent firing rates are allowed; (4) transitions can be assigned different priorities; (5) besides traditional output measures obtained from a GSPN, such as throughput of a transition and mean number of tokens in a place, more complex measures can be computed by using reward functions.

3. The VISION Cloud storage environment. In this section, we provide an overview of the storage cloud environment proposed by the VISION Cloud project [1] focusing on the implemented physical infrastructure and on the data model. We also provide details about GPFS-SNC [8], and about how it is used in the reference implementation of VISION Cloud.

3.1. The proposed storage cloud environment. The goal of the VISION Cloud project is to provide efficient support for data-intensive applications. Moreover, a content-centric view of storage services is provided. Five main areas of innovation drive the VISION Cloud platform design and implementation [10]: i) content is

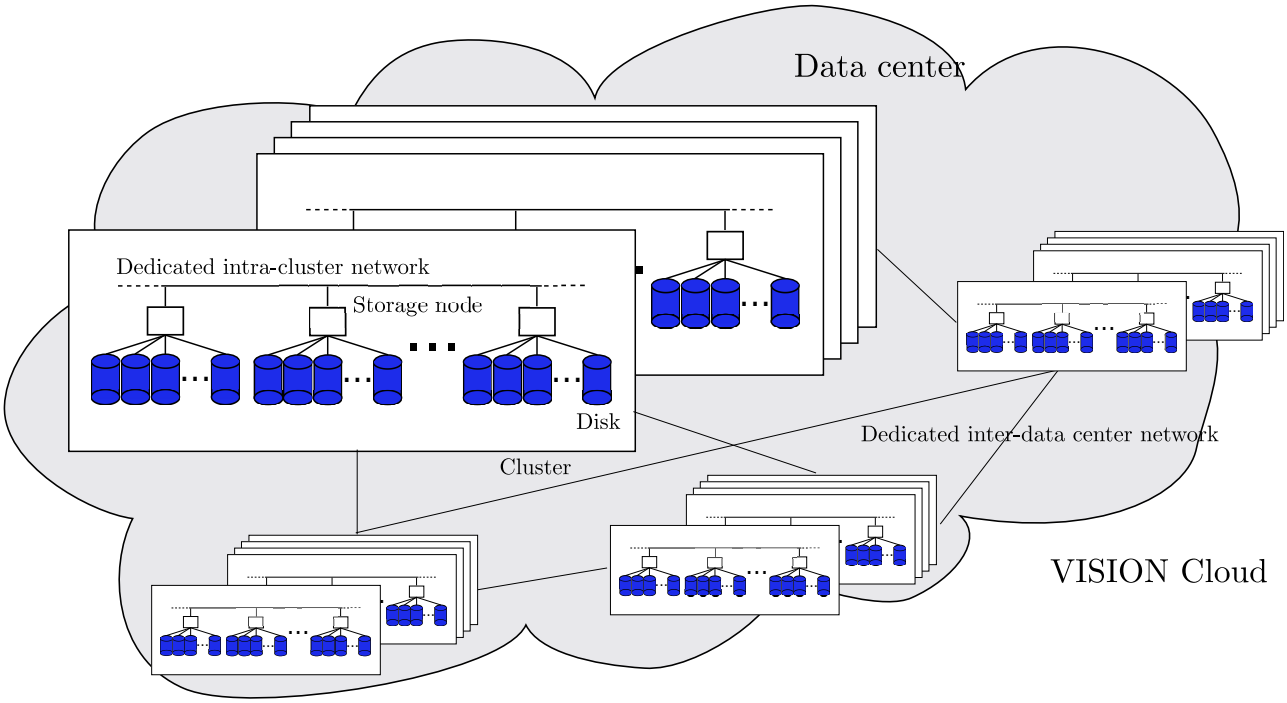


Fig. 3.1. *The VISION Cloud reference infrastructure.*

managed through data objects that can have rich metadata associated with them, ii) data lock-in is avoided by allowing migration of data across administrative domains, iii) computations are moved close to data through the use of storlets in order to avoid costly data transfers, iv) efficient retrieval of objects is allowed based on object content, properties, and relationships, and v) a high QoS level is guaranteed together with security and compliance with international regulations.

The storage cloud environment proposed by the VISION Cloud project is built on top of an infrastructure consisting of multiple data centers, potentially distributed worldwide. Each data center can be composed of one or more storage clusters containing physical resources providing computational, storage, and networking capabilities. The data centers need to be connected by dedicated high speed networks.

Each storage cluster is composed of storage rich nodes that can be built from commodity hardware and connected by commodity network devices. In fact, as common for cloud infrastructures, the storage cloud is built from low cost components and the desired reliability level is assured through the software layer. The software stack also builds advanced functionalities on top of this foundation. An example of initial hardware configuration could be 4 or 8 multiprocessor nodes with 12 to 16 GB of RAM each. Each node could have 12 to 24 high capacity direct attached disks (e.g., 2TB SATA drives). The architecture, design, and implementation of the VISION Cloud architecture supports a system with hundreds of storage clusters, where each storage cluster can have several hundred nodes and the storage clusters are spread out over dozens of data centers. Such a reference infrastructure is represented in Fig. 3.1.

The VISION Cloud data model is based on the concept of data object. A data object contains data of arbitrary type and size. It has a unique identifier that allows users to access it through the whole cloud. An object is written as a whole and cannot be partially updated, although it can be partially read. An object may be overwritten, in which case the whole content of the object is replaced. Versioning is supported. Data objects are stored in containers (with each data object residing within a single container). Containers provide easy data management, isolation, and placement policies. A rich metadata model allows system and user metadata to be associated with containers and objects. User metadata is set by the user and is transparent to cloud storage system. System metadata has concrete meaning to the cloud storage system.

The VISION Cloud data model extends traditional storage cloud models to include computation on the data objects, which is performed within the cloud storage environment through storlets. Storlets are software agents that are triggered according to specific events.

Objects may be replicated across multiple clusters and data centers. The degree of replication and placement restriction policies are defined and associated with an object's container. VISION Cloud employs a symmetric replication mechanism, where any operation on an object can be handled at any of its replicas. A storlet, when triggered, is executed once, usually at the site where the triggering condition first occurred.

3.2. GPFS-SNC as underlying distributed file system. In the storage cloud environment proposed by the VISION Cloud project, the simpler and lower level storage unit is the storage cluster. A distributed file system runs over the storage resources provided by each cluster (i.e., the servers and their direct attached disks). This allows each node to access the data objects stored in the cluster and to provide computational power on top of it by serving user requests and allowing the execution of storlets. In the current implementation of the VISION Cloud stack, the General Parallel File System for Shared Nothing Clusters (GPFS-SNC) is exploited in order to build such a distributed file system.

General Parallel File System (GPFS) [17] is a parallel file system for computer clusters providing the services of a general-purpose POSIX file system running on a single machine. GPFS supports fully parallel access to both file data and file system data structures (file system metadata). Moreover, administrative actions (e.g., adding or removing of disks) are also performed in parallel without affecting access to data. GPFS achieves its scalability through its shared storage architecture where all nodes in the cluster have access to all storage. Files are striped across all disks in the file system providing load balancing and high throughput. Large files are divided into equal sized blocks which are placed on different disks in a round-robin fashion. GPFS uses distributed locking to synchronize access to shared disks ensuring file system consistency while still allowing the necessary parallelism. As an alternative or a supplement to RAID, GPFS supports replication, storing two or more copies of each data or file system metadata block on different disks. Replication can be enabled separately for data and file system metadata.

The GPFS-SNC file system [8] builds on the existing GPFS distributed file system extending it to a shared-nothing cluster architecture. Such scenario is the one being used in the current implementation of VISION Cloud. In shared-nothing cluster architecture, every node has local disks behaving as primary server for them. If a node tries to access data and such a data is not present on a local disk, a request is sent to its primary server to transfer it.

In the reference implementation of VISION Cloud, each object is stored as a file in GPFS-SNC on a single disk. The files corresponding to objects are neither striped nor replicated within a cluster. Rather, additional object replicas are created in other VISION Cloud clusters in order to guarantee the desired level of availability. Typically a $(1+1, 1+1)$ schema is used for object replication, i.e., each object is replicated in two data centers at two storage clusters in each data center. However, other replication schema can be used changing the replication level. GPFS-SNC file system metadata is replicated with a certain level of redundancy in order to guarantee that the file system structure is preserved in the presence of faults and that it is possible to determine which object has been lost and needs to be recovered. The use of GPFS-SNC in the VISION Cloud architecture is graphically depicted in Fig. 3.2. In the remainder of the paper, we model a generic cluster file system with characteristics similar to those described above.

4. Problem formulation. In the following, we formally describe the scenario we take into consideration in the present work. Let us consider a VISION Cloud cluster composed by N nodes. Each node is associated with D directed attached storage (DAS) disks where both the distributed file system metadata and data (VISION Cloud objects) are stored. Note that, in the following we will consider only the distributed file system metadata (simply *metadata* from now on) while we ignore the system and user metadata associated with VISION Cloud objects, which are treated as files from the point of view of the cluster file system. Disks and nodes can fail. Let us suppose that the time to fail of a single disk (node) is exponentially distributed with rate λ_{df} (λ_{nf}). Disks (nodes) are repaired in an exponentially distributed time with rate μ_{dr} (μ_{nr}).

Disk and node failures are assumed to be destructive. In other words, when a disk fails the metadata and data stored in it are lost. Similarly, in order to maintain the distributed file system consistency, when a node fails metadata and data stored in all its attached disks are considered lost. VISION Cloud objects are stored

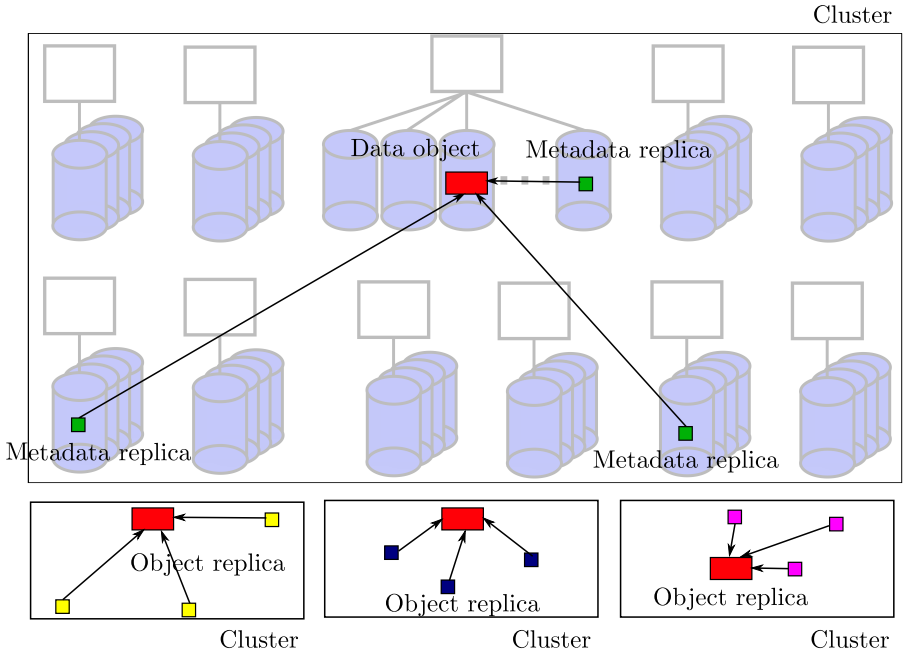


FIG. 3.2. *The use of GPFS-SNC in the VISION Cloud architecture.*

in the cluster without any striping or data replication, i.e., each object is fully contained in a single disk as a single file. On the other hand, metadata is scattered on the cluster disks and metadata records for each file are replicated on different nodes. Let us assume the level of metadata replication for each file to be R. This value is an internal parameter of the cluster file system, it is usually set during installation and it cannot be dynamically changed at runtime. When a disk fails the metadata that was present on it is replicated in a working disk in order to restore the correct level of replication. The process of metadata replication recovery takes an exponentially distributed amount of time with rate μ_{mr} to be performed.

VISION Cloud objects are replicated in other clusters. In the case of failure, the VISION Cloud Resiliency Manager (RM) is responsible for returning the storage Cloud to the proper level of resiliency. In fact, if a disk fails, a scan of the distributed file system metadata allows the RM to determine which objects were lost. Then, the RM contacts the other clusters in the Cloud (clusters in the same data center are usually queried first, since they are the closest) in order to recover the data from a replica and restore the objects into the cluster.

Let us consider a single VISION Cloud object X stored in the cluster. Objects are uniformly distributed over the cluster disks, i.e., when an object needs to be stored the target disk is randomly chosen accordingly to an uniform distribution. For such a reason, if a disk fails the probability that object X becomes unavailable (if it was still available at the failure time) is $1/x$ where x is the number of disks actually working with $0 < x \leq N \cdot D$. On the other hand, if a node fails the probability that object X becomes unavailable depends on the number of working disks that were attached to the failed node. In a first approximation, we assume that, given a VISION Cloud replication schema, at least one of the clusters in which object X was stored is always available for data recovery. Moreover, let us assume that, in order to recover an entire disk full of data, an exponentially distributed time is necessary with rate μ_{fd}. Among other factors, such a time can depend on the network bandwidth that is present between the consider cluster and the cluster from which the objects will be recovered.

Of course, given that the RM performs the data recovery as soon as possible after a disk failure, free space on other available disks is necessary in order to restore the lost objects in the cluster. Let us assume that the recovery can be performed only if there are at least K working disks in the local cluster. K can be computed considering the average disk capacity, the average object dimension, and the average number of objects in a cluster. For example, if c is the average fraction of occupied space in a disk then $K = \lceil N \cdot D \cdot c \rceil$. The time that is necessary to recover a single disk is also affected by the parameter c. In fact, the time needed to recover a

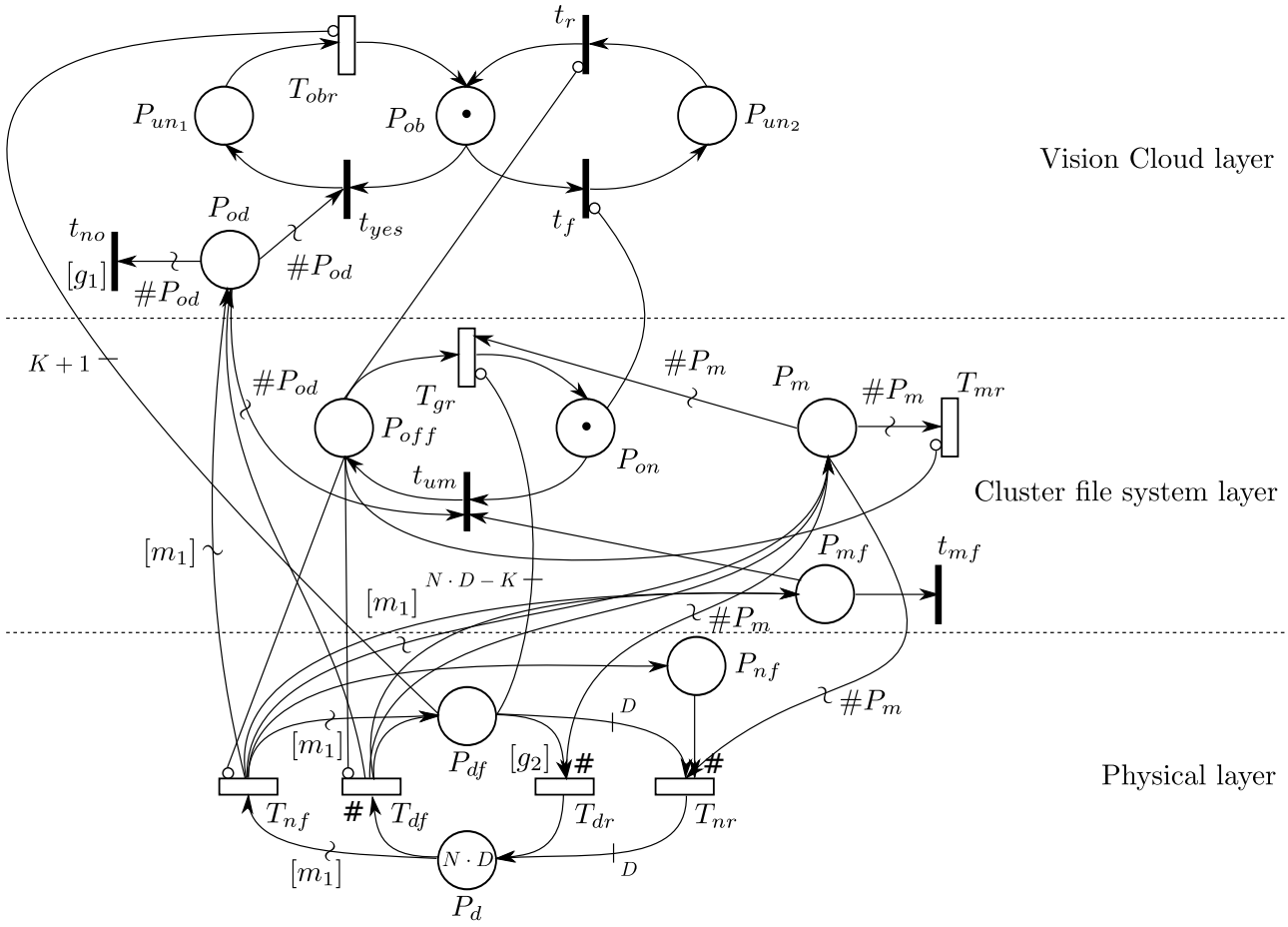


FIG. 5.1. *SRN model for a Vision Cloud cluster*

single disk (characterized by an average fraction of occupied space c) is considered as exponentially distributed with rate $\mu_{obr} = \mu_{fd}/c$.

Disk failures can affect the availability of a generic VISION Cloud object X even if it is not stored in the disk that fails. In fact, the distributed file system correctly works only until at least one metadata replica for each file is present on a working disk. If all the metadata replica for a single generic file are lost, the cluster file system is unmounted thus making object X unavailable. We suppose that when the cluster file system is unmounted no disk failures can occur and no objects recovery can be performed. The file system will be mounted again only when a sufficient number of disks are available again (we suppose such a sufficient number to be K). Moreover, all the objects that were originally present on the cluster need to be recovered. This is assumed to take an exponentially distributed time with rate μ_{gr} that can be computed as a function of μ_{fd}. In particular, if $N \cdot D$ disks are present in the cluster with an average fraction of occupied space c, then $\mu_{gr} = \mu_{fd}/(c \cdot K) = \mu_{fd}/(c \cdot \lceil N \cdot D \cdot c \rceil)$.

Note that a complete description of the VISION Cloud architecture and of its components is out of the scope of this paper. For further details please refer to [11].

5. The model. Figure 5.1 shows the SRN model for the Vision Cloud cluster described above. Three layers have been identified: physical layer (concerning node and disk failures and repairs), distributed file system layer (modeling the cluster file system metadata), and Vision Cloud layer (associated to object availability).

Places P_d and P_{df} represent working and failed disks, respectively. Place P_d initially contains $N \cdot D$ tokens while place P_{df} is initially empty. Each token represents a single disk. Transitions T_{df} and T_{dr} represent disk

failure and repair events moving tokens between places P_d and P_{df}. Rates of these transitions are considered to be dependent on the number of tokens in places P_d and P_{df}, respectively, so that the overall disk failure rate is equal to λ_{df} multiplied by the number of available disks while the overall repair rate is given by μ_{dr} multiplied by the number of failed disks. These marking dependent firing rates are represented by the # symbol near the corresponding arc.

Transitions T_{nf} and T_{nr} represent node failure and repair events. The failure of a single node is modeled as the contemporaneous failure of more than one disk by letting transition T_{nf} to move more than one token from place P_d to place P_{df}. This is obtained by associating to the arcs connecting transition T_{nf} to places P_d and P_{df} a multiplicity that depends on the actual status of the net through function $[m_1]$. In particular, the number of disks that contemporaneously fail when a node fails is assumed to be dependent on the actual number of failed nodes and disks: if nf nodes and df disks are failed, then we assume that the average number of disks that fail when a node fails is given by $(N \cdot D - df)/(N - nf)$. Considering that transition T_{nf} also puts a token in place P_{nf} at each node failure event (i.e., tokens in place P_{nf} model the number of failed nodes), we have:

$$[m_1] = \#P_d/(N - \#P_{nf})^{\dagger}.$$

The rate of transition T_{nf} also depends on the actual status of the net and, in particular, it is equal to λ_{nf} multiplied by the number of working nodes, i.e., $\lambda_{nf} \cdot (N - \#P_{nf})$. The repair of a single node is modeled as the contemporaneous repair of D disks. For such a reason, each firing of transition T_{nr} moves D tokens from place P_{df} to place P_d. Also, one token is removed from place P_{nf} in order to model a single node being repaired. The rate of transition T_{nr} depends on the number of tokens in place P_{nf} so that the overall node repair rate is equal to λ_{nr} multiplied by the number of failed nodes.

Finally, transition T_{dr} is associated with guard function $[g_2]$ that allows single disks to be repaired only if there is a sufficient number of working nodes:

$$[g_2] = \begin{cases} 1, & \text{if } \#P_{df} > D \cdot \#P_{nf} \\ 0, & \text{otherwise} \end{cases}$$

In this way, if all the failed disks correspond to failed nodes, transition T_{dr} is disabled.

Place P_m represents failed metadata replicas that need to be restored. It initially contains zero tokens. As soon as a disk fails (transition T_{df} fires) or a node fails (transition T_{nf} fires), a number of tokens equal to the number of failed disks is moved in place P_m representing the corresponding metadata replicas being lost. Transition T_{mr} represents the time necessary for the failed metadata replicas to be restored on the cluster. It is associated with a rate equal to μ_{mr} and, as soon as it fires, it flushes the content of place P_m modeling all the metadata replicas being restored. This is implemented by associating to the arc connecting transition T_{mr} to place P_m a multiplicity equal to the number of tokens in such a place.

As soon as a certain number of disks fail (either transition T_{df} or transition T_{nf} fires), a token is also put in place P_{mf} enabling the conflicting immediate transitions t_{mf} and t_{um}. Transition t_{mf} models the probability for the cluster file system to continue to work properly after the newly occurred failure conditioned to the fact that it was correctly working when the failure occurred. Such a probability depends on the actual number of working nodes and metadata replicas present in the cluster so it can be computed as a function of the current number of tokens in places P_d and P_m. As soon as transition t_{mf} fires, it removes the token from place P_{mf} leaving everything else unmodified. On the other hand, transition t_{um} models the probability for the cluster file system to be unmounted after the newly occurred failure conditioned to the fact that it was correctly working when the failure occurred. Also in this case, such a probability depends on the actual number of working nodes and metadata replicas present in the cluster and it can be computed as a function of the current number of tokens in places P_d and P_m. Given that transitions t_{mf} and t_{um} are conflicting and no other transition is contemporaneously enabled the sum of their associated probabilities needs to be equal to one. As soon as transition t_{mf} fires, a token is moved from place P_{on} to place P_{off}. Moreover, the token in place P_{mf} is removed.

Place P_{on} represents a working distributed file system while place P_{off} represents a faulty file system. When the cluster file system is down, no new metadata replica can be created (inhibitor arc from place P_{off} to transition T_{mr}) and no disks or nodes can fail (inhibitor arcs from place P_{off} to transitions T_{df} and T_{nf}).

Transition T_{gr} represents the time necessary to repair the distributed file system after a crash due to

†The notation $\#P$ indicates the number of tokens in place P.

metadata destruction, to recover all the objects from the replicas in other Vision Cloud clusters, and to create the metadata replicas. It is associated with a rate equal to μ_{gr}. Such recovery operation can be performed only after the repair of at least K disks (inhibitor arc from place P_{df} to transition T_{gr} with multiplicity $N \cdot D - K$). As soon as transition T_{gr} fires, a token is put back to place P_{on} (the cluster file system is up again) and all the tokens in place P_m are flushed modeling the recovery of all the failed metadata replicas. This is implemented by associating to the arc connecting transition T_{gr} to place P_m a multiplicity equal to the number of tokens in such a place.

A token in place P_{ob} represents the object being available. As soon as a failure occurs, a number of tokens equal to the number of failed disks is moved in place P_{od} by transitions T_{df} or T_{nf}. Such tokens enable the conflict between transitions t_{yes} and t_{no} representing the object being contained in the disks that failed or not, respectively. The probabilities associated to transitions t_{yes} and t_{no} (p_{yes} and p_{no}, respectively) depend on the system status and are given by the following functions:

$$p_{yes} = 1/(\#P_d + \#P_{od})$$

$$p_{no} = \begin{cases} 1, & \text{if } \#P_d = 0 \text{ AND } \#P_{un_1} = 1 \\ 1 - 1/(\#P_d + \#P_{od}), & \text{otherwise} \end{cases}$$

Transition t_{no} is also associated with a guard function ($[g_1]$) that prevents it to fire if the last disk failed:

$$[g_1] = \begin{cases} 0, & \text{if } \#P_d = 0 \text{ AND } \#P_{ob} = 1 \\ 1, & \text{otherwise} \end{cases}$$

If transition t_{no} fires, the object was not contained in the disks that failed and it is still available. If transitions t_{yes} fires, the object was contained in one of the disks that failed and the token in place P_{ob} is moved in place P_{un_1} modeling the object being unavailable. Transition T_{obr} represents the time necessary to recover the object from another Vision Cloud cluster where a replica of that object is present. It is associated with a rate equal to μ_{obr}. The recovery operation can be performed only when at least K disks are available (inhibitor arc from place P_{df} to transition T_{obr}). The token in place P_{ob} can also be moved in place P_{un_2} when the cluster file system is unmounted for a metadata destruction (transition t_f). A soon as the cluster file system is repaired, transition t_r fires and the object becomes available again.

5.1. Cluster file system failure probability. In order to properly set the model parameters (i.e., the probabilities associated to transitions t_{mf} and t_{um}) we need to know the probability that the cluster file system is unmounted when a new failure condition arises. Such a probability depends on the number of metadata replica as well as on the way the replica are distributed over the disks and the nodes. The problem can be formalized in the following way.

Let us start by defining a working condition where:

- n ($\leq N$) is the number of actual working nodes.
- d ($\leq D$) is the average number of working disks per node.

Indicating with MF the total number of metadata records, we are interested in the evaluation of the following probability:

- $P^{n,d,R,MF}(i)$ = Probability that, in the working condition defined by the pair (n, d), there is still one (out of the R) copy of each of the MF metadata files after i disk failures, given that the system was still working before the last failure

subjected to the following constrains:

1. Metadata are not restored.
2. If a node fails all its disk have to be considered failed, i.e., we have to consider the concurrent failure of d disks.
3. Replica are distributed so that, as long as there is a sufficient number of working nodes (i.e., $n \geq R$), two copies of the same file are not stored in the same node.

An estimation of MF can be obtained by considering the number of VISION Cloud objects actually stored in the cluster O and the number of metadata replica R as

$$MF = 1.1 \cdot O \cdot R \tag{5.1}$$

where the factor 1.1 refers to the assumption of a 10% overhead due to directory structure and VISION Cloud user and system metadata. The analytical solution of such a problem is intractable for large-scale systems [20], for this reason we solved the problem through simulation. We set up a simple simulator that starting from the values n, d, R, and MF creates a scenario by distributing metadata in an uniform way (still taking into account the constrains). Then, we iteratively introduce a failure (also in this case using an uniform distribution) until a distributed file system fault is encountered.

6. Results. The SRN model reported in Fig. 5.1 can be analytically solved by using ad-hoc tools (e.g., the SPNP tool [7]) thus allowing us to investigate the influence of system parameters on the desired performance indexes. Several powerful measures can be obtained. One interesting index is the availability A_{ob} of a generic object X formally defined as the probability that the object is fully accessible from external users at steady state. It can be obtained by computing the probability for place P_{ob} to contain one token:

$$A_{ob} = pr[\#P_{ob} = 1].$$

Similarly, the cluster availability A_{cl} (formally defined as the probability that the cluster file system is properly working at steady state) can be computed as the probability for place P_{on} to contain one token:

$$A_{cl} = pr[\#P_{on} = 1].$$

In this section, we present some preliminary results focusing on the object availability and taking into account only disk failures (i.e., considering fully reliable nodes). The relaxation of such an assumption, as well as the investigation of other performance indexes will be covered in future works.

System parameters have been set as follows. The number of nodes N has been fixed to 80 and the number of disks per node D has been fixed to 12, also considering the average fraction of occupied space in a disk c equal to 0.5. The disk mean time to failure (MTTF) $1/\lambda_{df}$ has been considered equal to 2 *years* [18] while the mean time to repair (MTTR) $1/\mu_{dr}$ has been set to 48 h. Finally, the mean time to recover a metadata replica $1/\mu_{mr}$ has been set to 20 m. The mean time to recovery an entire disk from a remote cluster has been computed by assuming the disk dimension equal to 500 GB and by examining different scenarios with different level of bandwidth among the clusters in the same Vision Cloud. Three scenarios have been considered: HPC-like connectivity (10 Gb/sec bandwidth), high-speed WAN connectivity (100 Mb/sec bandwidth), and Internet-like connectivity (20 Mb/sec bandwidth). In the following, the three scenarios will be identified as *high*, *medium*, and *low* bandwidth scenario, respectively. Starting from the above reported assumptions, the values of the mean time to recover a disk $1/\mu_{obr}$ and the mean time to recover an entire cluster file-system $1/\mu_{gr}$ have been computed, as described in Section 4.

In the first experiment, we aim to investigate the influence of the metadata replication level R. First of all, in order to obtain the values of $P^{n,d,R,MF}(i)$ in all the working conditions, once the values for N and D have been chosen, we launched the simulator whit $n = 1, ..., N$ and $d = 1, ..., D$. The value of MF has been estimated[‡] through Eq. (5.1) by considering an average object size equal to $8MB$ (that can be considered a realistic example considering the presence of different kind of file, e.g, audio, photo, document, video files) and a corresponding number of object $O = \frac{500GB \cdot N \cdot D \cdot c}{8MB} = 30,720,000$. Data obtained for different value of R have been then collected in a file that has been used during the evaluation of the SRN model. Figure 6.1 shows the results obtained with $n = 80$, $d = 12$ and varying R from 3 to 5 (such values of R can be considered a good trade-off between redundancy and storage consumption). It can be observed that, as expected, the distributed file system failure probability increases when the number i of failed disks increases, reaching a value near to 1 when $i = 7, 22, 47$ with $R = 3, 4, 5$ respectively. Such a result highlights the influence of the replication level on the system fault tolerance. However in order to quantify the advantages obtained in terms of object availability, we solved the model using the $P^{n,d,R,MF}(i)$ values as input thus obtaining the data reported in Table 6.1. These data refer to the values of A_{ob} obtained in the three bandwidth scenarios. It can be observed that the influence of R strictly depends on the network bandwidth. In fact, in the low bandwidth scenario the object availability increases from a value of 0.95 to a value of 0.99 when R changes from 3 to 5, with a percentage gain of about 4%. On the contrary, in the high bandwidth scenario we obtain, in the same conditions, only a percentage gain of about 0.009%.

[‡]In the computation of the value of O the storage space occupied by metadata has been neglected.

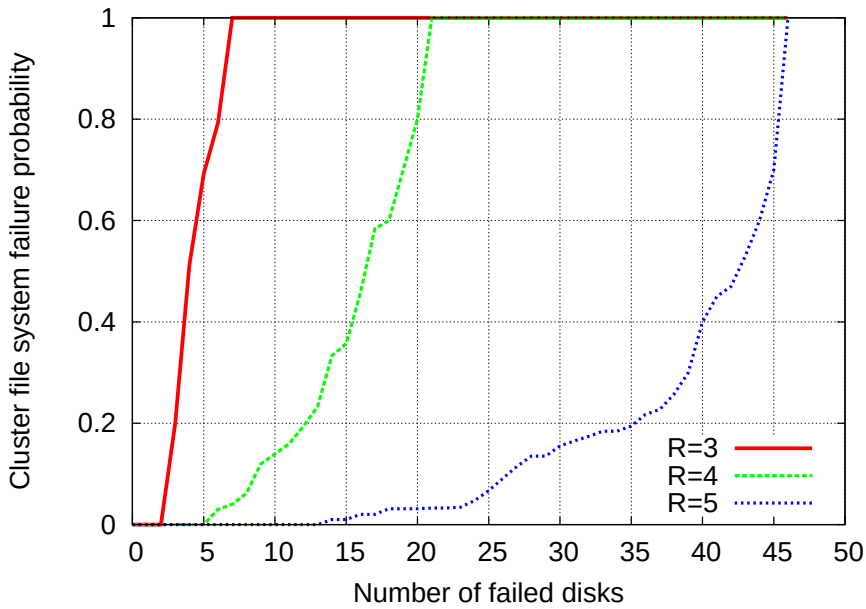


FIG. 6.1. *Cluster file system failure probability with respect to the number of disk failures ($n = 80$, $d = 12$, $MF = 30,720,000 \cdot 1.1 \cdot R$).*

	Low	Medium	High
$R = 3$	0.9548245227	0.9906377221	0.9999077434
$R = 4$	0.9983792830	0.9996754359	0.9999968294
$R = 5$	0.9983795900	0.9996754973	0.9999968300

TABLE 6.1
Object availability A_{ob} varying R in three different bandwidth scenarios.

In the next experiment, we focus on the influence of the disk MTTF. Figure 6.2 shows the results obtained varying the value of the MTTF from 500 to 900 days in the medium bandwidth scenario. Such an analysis shows how increasing the disk MTTF it is possible to increment the overall object availability from a user perspective. This can be performed by choosing more reliable disks or exploiting RAID technologies with a consequent increase in the operating costs of the storage cloud infrastructure.

The above reported results give rise to interesting optimization problems. In fact, the Vision Cloud provider could take advantages of the proposed model in order to obtain useful insights during the design of a Cloud infrastructure. For example, given a certain level of desired availability and given the topological configuration of the clusters (in terms of network bandwidth), the provider can use the models to obtain the optimal number of replica to adopt and the needed disk reliability (in terms of MTTF). Similarly, per-user model-driven design could be conducted in order to optimize the placement of objects: according to the availability level requested by a single user the Cloud provider can understand in which clusters the user object replica have to be stored.

7. Conclusions. In the context of the VISION Cloud project reference architecture, we provided an SRN model for a storage cluster able to provide information about the reached availability level. Numerical results demonstrated the effectiveness of the proposed model. In fact, the model can be exploited for an assisted SLA management and a guided dimensioning of the VISION infrastructure. Future work will focus on extending the obtained results to the case of node failures and relaxing the simplifying hypothesis that we took into consideration in the present work, e.g., considering transient failures that can affect the overall object and cluster availability. Moreover, a high level methodology and a tool for the management of VISION Cloud

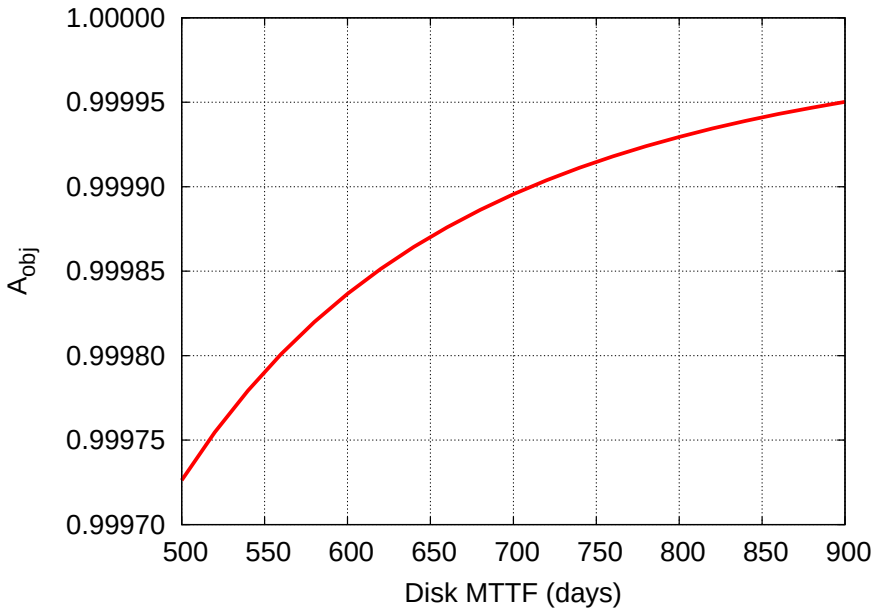


FIG. 6.2. *Object availability A_{ob} varying the disk MTTF in a medium bandwidth scenario with $R = 3$.*

storage infrastructures based on our model will be designed and implemented providing a powerful tool for both business and administrator choices. Finally, comparison of the obtained results against real world observation will be carried out in order to validate the model.

Acknowledgement. The research leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement number 257019.

REFERENCES

[1] *VISION Cloud Project, funded by the European Commission Seventh Framework Programme (FP7/2006-2013) under grant agreement n. 257019.* http://www.visioncloud.eu/.

[2] D. Bruneo, *A stochastic model to investigate data center performance and qos in iaas cloud computing systems*, Parallel and Distributed Systems, IEEE Transactions on, PP (2013), pp. 1–10.

[3] D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and M. Scarpa, *Workload-based software rejuvenation in cloud systems*, IEEE Transactions on Computers, 62 (2013), pp. 1072–1085.

[4] D. Bruneo, M. Fazio, F. Longo, and A. Puliafito, *Smart data centers for green clouds*, in Computer and Communications (ISCC), 2013 IEEE 18th International Symposium on, 2013, pp. 1–8.

[5] D. Bruneo, M. Scarpa, and A. Puliafito, *Performance evaluation of glite grids through gspns*, Parallel and Distributed Systems, IEEE Transactions on, 21 (2010), pp. 1611–1625.

[6] G. Ciardo, A. Blakemore, P. F. Chimento, J. K. Muppala, and K. S. Trivedi, *Automated generation and analysis of Markov reward models using stochastic reward nets.*, IMA Volumes in Mathematics and its Applications: Linear Algebra, Markov Chains, and Queueing Models, 48 (1993), pp. 145–191.

[7] C. Hirel, B. Tuffin, and K. S. Trivedi, *SPNP: Stochastic Petri Nets. Version 6*, in International Conference on Computer Performance Evaluation: Modelling Techniques and Tools (TOOLS 2000), B. Haverkort, H. Bohnenkamp (eds.), Lecture Notes in Computer Science 1786, Springer Verlag, Schaumburg, IL, 2000, pp. 354 – 357.

[8] R. Jain, P. Sarkar, and D. Subhraveti, *Gpfs-snc: An enterprise cluster file system for big data*, IBM Journal of Research and Development, 57 (2013), pp. 5:1–5:10.

[9] A. Khan, X. Yan, S. Tao, and N. Anerousis, *Workload characterization and prediction in the cloud: A multiple time series approach*, in Network Operations and Management Symposium (NOMS), 2012 IEEE, 2012, pp. 1287–1294.

[10] E. Kolodner, S. Tal, D. Kyriazis, D. Naor, M. Allalouf, L. Bonelli, P. Brand, A. Eckert, E. Elmroth, S. Gogouvitis, D. Harnik, F. Hernandez, M. Jaeger, E. Lakew, J. Lopez, M. Lorenz, A. Messina, A. Shulman-Peleg, R. Talyansky, A. Voulodimos, and Y. Wolfsthal, *A cloud environment for data-intensive storage services*, in Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, 2011, pp. 357–366.

[11] E. K. Kolodner, A. Shulman-Peleg, D. Naor, P. Brand, M. Dao, A. Eckert, S. Gogouvitis, D. Harnik, M. Jaeger, D. Kyriazis, et al., *Data intensive storage services on clouds: Limitations, challenges and enablers*, European Research Activities in Cloud Computing, D. Petcu and JL Vazquez-Poletti, Eds. Cambridge Scholars Publishing, (2012), pp. 68–96.

[12] S. Krishnamurthy, W. Sanders, and M. Cukier, *A dynamic replica selection algorithm for tolerating timing faults*, in Dependable Systems and Networks, 2001. DSN 2001. International Conference on, 2001, pp. 107–116.

[13] ———, *Performance evaluation of a probabilistic replica selection algorithm*, in Object-Oriented Real-Time Dependable Systems, 2002. (WORDS 2002). Proceedings of the Seventh International Workshop on, 2002, pp. 119–127.

[14] M. A. Marsan, G. Balbo, and G. Conte, *A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems*, ACM Transactions on Computer Systems, 2 (1984), pp. 93–122.

[15] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, *A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing*, in Cloud Computing, vol. 34 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, ch. 9, pp. 115–131.

[16] C. Petri, *KommuniKation mit Automaten*, PhD thesis, University of Bonn. Germany, 1962.

[17] F. Schmuck and R. Haskin, *Gpfs: A shared-disk file system for large computing clusters*, in In Proceedings of the 2002 Conference on File and Storage Technologies (FAST, 2002, pp. 231–244.

[18] B. Schroeder and G. A. Gibson, *Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you?*, in Proceedings of the 5th USENIX Conference on File and Storage Technologies, FAST '07, Berkeley, CA, USA, 2007, USENIX Association.

[19] V. Venkatesan, I. Iliadis, C. Fragouli, and R. Urbanke, *Reliability of clustered vs. declustered replica placement in data storage systems*, in Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on, 2011, pp. 307–317.

[20] V. Venkatesan, I. Iliadis, X.-Y. Hu, R. Haas, and C. Fragouli, *Effect of replica placement on the reliability of large-scale data storage systems*, in Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on, 2010, pp. 79–88.

Edited by: Maria Fazio and Nik Bessis
Received: Nov 2, 2013
Accepted: Jan 10, 2014

DOI 10.12694/scpe.v14i4.934
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. 291–306. http://www.scpe.org

DELEGATION ACROSS STORAGE CLOUDS: ON-BOARDING FEDERATION AS A CASE STUDY

CIRO FORMISANO[3] AND ELLIOT K. KOLODNER[2] AND ALEXANDRA SHULMAN-PELEG[2] AND ERMANNO TRAVAGLINO[3] AND GIL VERNIK[2] AND MASSIMO VILLARI[1] *

Abstract. As the volume of digital data rapidly increases, storage clouds are becoming a popular solution for both enterprise and personal data, and the number of storage cloud solutions is also increasing. However, these solutions do not yet deal with the need of customers for interoperability and data migration from one cloud to another. These issues can be addressed through federation of cloud infrastructures. An important aspect of federation is delegation of access control, where one actor, e.g., an end user, authorizes another actor, e.g., a cloud provider, to act on its behalf, typically with a subset of its access rights, safely and securely.

This paper deals with delegation across storage clouds. We describe a delegation architecture for on-boarding federation, which allows an enterprise to efficiently migrate its data from one storage cloud provider to another (e.g., for business or legal reasons), while providing continuous access and a unified view over the data during the migration. In our architecture a user delegates a subset of his access rights on the source and destination clouds to an on-boarding federation layer on the destination cloud. This enables on-boarding to occur in a safe and secure way, such that the on-boarding layer has the least privilege required to carry out its work. We evaluate the security implications of delegation that need to be taken into account for on-boarding. We also show how the delegation architecture can be implemented using the Security Assertion Markup Language.

Key words: Storage Cloud, Federation, Delegation, SAML.

1. Introduction. Existing storage clouds do not provide true data mobility as well as adequate mechanisms for allowing efficient migration of their data across providers. This of problem of "data lock-in" is considered to one of the top ten obstacles for growth in Cloud Computing [1]. In a recent paper Vernik et al. [30] present an architecture for *on-boarding federation* to deal with this problem. On-boarding federation allows an enterprise to efficiently migrate its data from one storage cloud provider to another (e.g., for business or legal reasons), while providing continuous access and a unified view over the data over the course of the migration. On-boarding is provided through a federation layer on the new destination cloud by setting up a relationship between its containers and the containers on the old source cloud. Once the relationship is set up, the on-boarding layer is responsible to carry out the migration on behalf of the user, reading objects from the old source cloud and writing objects to the new destination cloud. This layer acts on behalf of the user and requires authorization from the user to act in his/her name with the old and new providers.

A *delegation* mechanism empowers one actor, e.g., an end user, to authorize another actor, e.g., a cloud provider, to act on its behalf, typically with a subset of its access rights, safely and securely. In this paper we show how to employ delegation for on-boarding federation. In particular, when a user sets up an on-boarding relationship between a container in the new and old clouds, the user also delegates a subset of his/her access rights to the federation layer of the new cloud. This subset should include the minimum rights needed for the federation layer to on-board objects of the old container. The delegation mechanism is also secure; it ensures that no other entity except the federation layer can employ the rights delegated to it.

In this paper, we consider two popular standards for delegation, OAuth 2.0 [28] and the Security Assertion Markup Language (SAML) [24]. Motivated by the better security of SAML, we developed an architecture allowing to use it for delegation between the two clouds involved in on-boarding federation. We detail our solution implemented in the context of the VISION Cloud [13], which is an EU-funded project developing advanced features for storage clouds. We provide details of the delegation API and the SAML assertions used to implement it.

The paper is organized as follows. Section 2 overviews the VISION Cloud architecture and Sect. 3 describes the principles of cloud access control and delegation. Section. 4 describes the difference between single sign-on and delegation, and then presents and compares two techniques for web delegation, OAuth 2.0 and SAML. Sect. 5 presents use cases for delegation in VISION Cloud. We present our delegation architecture for on-

*1)Dept. of DICIEMA, University of Messina, Italy 2)IBM Haifa Research Lab, Israel 3)Engineering Ingegneria Informatica SPA, Italy

boarding and its implementation using SAML in Sect. 6. Sect.s 7 and 8 review the related work and conclude. Finally, the appendices detail the delegation API and the SAML assertions implementing it.

2. A brief overview of VISION Cloud. The VISION Cloud architecture is designed to support tens of geographically dispersed data centers (DCs, see Figure 2.1), where each DC may contain tens of clusters each with hundreds of storage-rich compute nodes. An object consists of data and metadata, where the data can be of arbitrary size and type, and the metadata can include arbitrary key value pairs, typically describing the content of the object. Object data cannot be updated, but an entire new version of an object can be created.

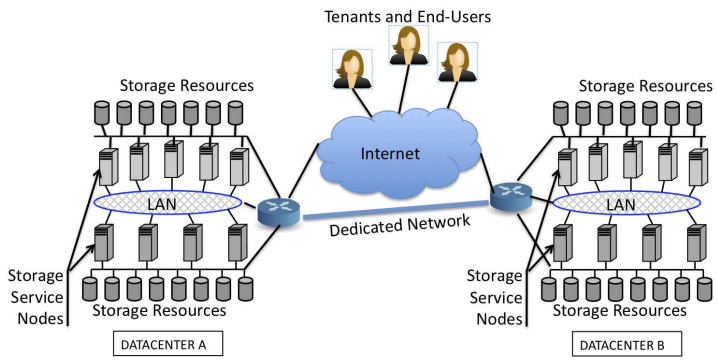


FIG. 2.1. *VISION Cloud Reference Framework: the physical view*

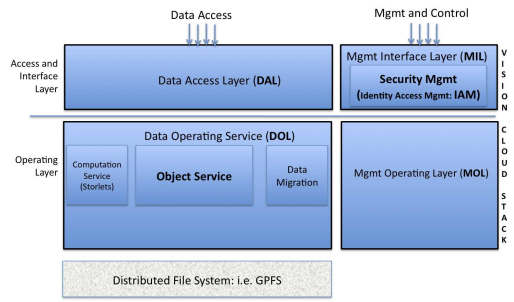


FIG. 2.2. *Simplified Stack of VISION Cloud: the logical view*

In contrast, new metadata can be appended to an object and updated over time. Data objects are grouped in containers, and each data object has a unique name within a container. Containers provide context and are used for purposes of data management, isolation, and placement (i.e., containers are the minimal units of placement and can not be split across clusters). Metadata can also be associated with containers.

The account model includes tenants and users. A tenant subscribes to cloud storage services and usually represents a company or an organization. A tenant may have many users. A tenant administrator creates user accounts and manages them. A user has an identifier and may have credentials allowing him to authenticate himself to the cloud; finally, a user may create containers and data objects in them.

A user is the entity that actually uses storage services, and may refer to a person or to an application. A user belongs to one and only one tenant, although a person might own a user account in more than one tenant.

Figure 2.2 shows a simplified view of the VISION Cloud stack. On the left side, the *Data* layers are depicted wheres on the right side there are represented the *Management* layers. The picture highlights the two main elements (Object Service and Identity Access Manager, see the text of labels in bold) involved in this paper.

3. Identity and access management systems for storage clouds. Most storage clouds provide RESTful interfaces to allow manipulation of resources with standard HTTP methods. Thus, we address access control methodologies in the context of web technologies, considering two stages: authentication and authorization. Authentication is the phase responsible for verifying the identity of a user needing to access web resources. Authorization is the phase responsible for verifying the operation (i.e., read, write, delete, etc.) that users may make on web resources. We use the term **Identity and Access Management (IAM)** to identify the systems able to perform the phases mentioned above. In federated storage cloud scenarios the IAM systems may assume more complex configurations. There are three identity management architectures that are relevant for federation:

- **One Shared IAM:** In this set up the two federated storage clouds use the same internal IAM. This architecture is possible when there are two Clouds deployed on the same public infrastructure. For example, it could correspond to two different OpenStack Swift [27] deployments sharing the same Keystone [10] infrastructure. Any authentication protocol can be used in this set up.
- **One External IAM:** In this case each cloud customer (tenant) has its own IAM, defining an *identity and access domain* that establishes a trust relationship with both federated Clouds. This architecture

allows the user to authenticate with a single set of credentials for applications residing at different cloud and non-cloud systems.

- **Two IAMs:** Each cloud may have its own internal IAM. For example, let's consider two clouds named A and B, in which A needs to access resources hosted in B. In this case the Cloud A should have the credentials on IAM of B to authenticate against the Cloud B.

In the case of *One IAM* (shared or external), the access control is not particularly complex, because the two clouds rely on the same IAM. However, in the case of *Two IAMs* the scenario is much more complex. Every cloud, having an internal IAM, needs to overcome the issues of authentication and authorization of users belonging to different administration domains. Delegation is a common way of overcoming the problems while preserving the proper privileges of users of each cloud. Before discussing our solution to the problem of two IAMs, we describe the concept of delegation (see Sect. 3.1) and compare the existing web delegation technologies (see Sect. 4).

3.1. Introduction to Delegation. Delegation provides the capability for a user (*U1*) to delegate a subset of his access privileges to another user or process (*U2*). *U1* is called *delegator* while *U2* is the *delegate*. Both the users keep their own identities, but *U2* obtains a delegation document signed by *U1* and stating that *U2* is authorized to act on behalf of *U1* for certain operations. Note that *U2* does not obtain the identity of *U1*, i.e., *U2* **does not impersonate** *U1*, rather *U2* is explicitly authorized to perform certain actions on behalf of *U1*. An example taken from everyday life, would be a person going to a public office to get a document on behalf of another person. In this case the human delegate shows his own ID card and a *Power of Attorney*, the delegation document, signed by the delegator, and a copy of the delegator's ID card.

In computer science the concept is identical, in particular:

- The *delegator* provides the *delegate* with an electronic delegation document, digitally signed, containing the details of the delegation (permitted operations, possibility to transfer the delegation, etc.)
- The *delegate* provides his credentials and the delegation document in order to use the delegation and obtain access.

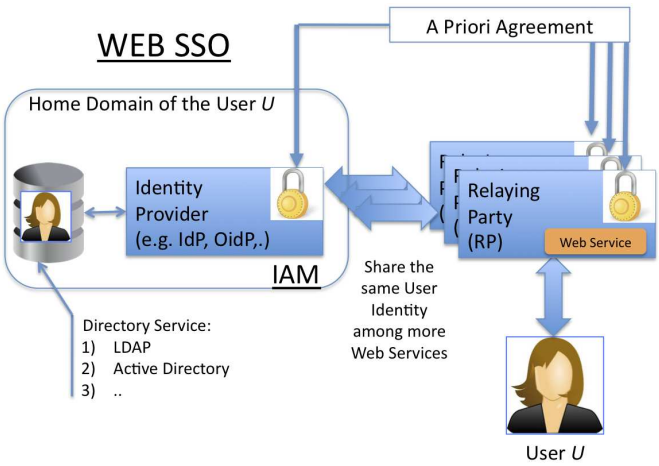


FIG. 4.1. *An example showing Web Single Sign-on*

4. Web Single Sign-on vs. Access Delegation. Single Sign-On (SSO) is an identity federation mechanism removing the need to have multiple accounts on different Relying Parties. Other terms for this party include Service Provider (*SP*). The user can login once to one provider and then access multiple web services. Figure 4.1 shows a typical SSO system, consisting of a *Relying Party* (*RP*), an *IAM* (Identity Provider or OpenID Provider - *IdP*, *OIdP - relying on an internal Directory Service*), and the *end-user* (here identified with the User U). User U has an identity registered in the IAM and needs to access a certain web resource on RP:

1. The user is redirected to IAM for the authentication (the authentication **must** be performed on the home domain).
2. If the authentication has succeeded, he/she is redirected to RP for obtaining the requested resource.

Examples of protocols providing Web SSO are SAML [24], OpenID [20], CAS [2]. SAML, OpenID and CAS work at different levels: the first two protocols provide inter-domain Identity Federation (with some difference), while CAS provides only SSO on a single domain.

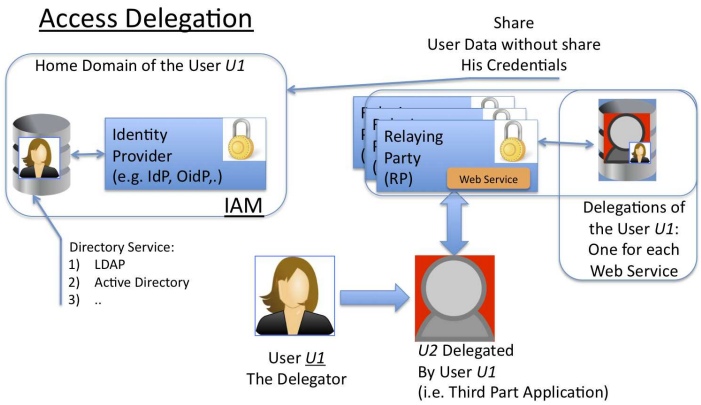


FIG. 4.2. *An example showing the Access Delegation procedure*

The Access Delegation procedure is conceptually different than identity federation in SSO. It is the mechanism employed when a consumer application (e.g., Instagram) wants to use Twitter(Tw) or Facebook (Fb) to post a Tweet or write on a Facebook wall on behalf of an end user. Let's assume an *end-user*, that after downloading a consumer application from mobile market place, decides to exploit the application for interacting with Tw/Fb resources. At the early stage the user gains the access to Tw/Fb through the consumer application, hence the application itself will continue to work without any user supervision. Specifically, the user gains temporary access to Fb/Tw, obtaining *special* tokens necessary for the consumer application. These tokens are uniquely bound and tailored to the application, and they can allow limited operations on behalf of the user. Figure 4.2 depicts all parts composing a typical Access Delegation system. In the figure the Service Provider (SP) can be Fb and/or Tw services, whereas the Third Part Application is equivalent to a consumer application (e.g., Instagram, see the icon avatar in the figure.) For example, using this delegation model the Instagram application is able to post pictures on Facebook on behalf of the end-user. To summarize:

- Web SSO provides a mechanism to *Share the same User Identity* among multiple Web Services. Hence, SSO allows logging in once and then accessing several web services.
- Access Delegation provides a mechanism to *Share User Resources* without sharing credentials.

Access delegation has been widely used in multiple applications implemented with well know protocols like OAuth [28]. However, when considering a setup with delegation across two clouds OAuth protocols have the limitation that they assume the existence of an IAM server shared between the clouds. Another way to setup delegation is using a modified version of the SAML protocol. SAML was originally designed for allowing SSO, but with several new specifications it is now able to provide delegation as well, e.g., through SAML 2.0 Condition to Delegate [22]. The next section introduces OAuth and SAML 2.0 Condition to Delegate solutions, highlighting the advantages and the disadvantages of each.

4.1. OAuth 2.0. The RFC 6749, defined by the IETF, describes the OAuth 2.0 Authorization Framework, and it specializes the existing features of OAuth 1.0 (for further details see the reference [7]). In particular the OAuth 2.0 Authorization Framework allows a third-party application to achieve limited access to an HTTP service. Looking at the OAuth 2.0 Framework it is possible to identify the following actors: *Client*, *Resource Owner*, *Authorization Server*, and *Resource Server*. The **Client** is the application that needs to access protected resources on behalf of the user **Resource Owner** that is the user owning the protected resources that the client needs to access. **Authorization Server** is the server that provides access tokens to the client after the user

has been authenticated and has been granted access to his resources. **Resource Server** is the server managing the user protected resources.

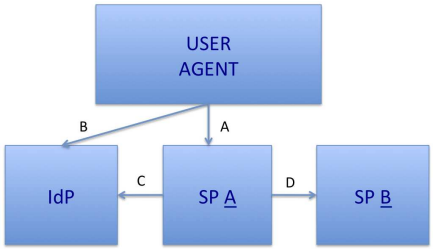


FIG. 4.3. *SAML Condition to Delegate Abstract Protocol Flow*

4.2. SAML 2.0 Condition to Delegate. SAML was introduced by the OASIS consortium following the standardization of XML docs (i.e., SAML [24], XACML [31]). The SAML 2.0 Condition to Delegate (SAML2Del) specification [22] shows how to use the SAML 2.0 protocol for delegation. In particular SAML2Del describes the following delegation-like scenarios:

- **Proxying** where an intermediate identity provider issues an assertion to a relying party on the basis of an assertion issued to it. Proxying is a gateway in which the subject of the assertion is presumed to directly interact with each party.
- **Impersonation** where an entity acting on behalf of an assertion subject is able to obtain and use an assertion indistinguishable from an assertion that can be issued directly to the subject.
- **Forwarding** where an assertion is directly reused by an intermediary to impersonate a subject from whom an assertion was obtained. Forwarding is a form of Impersonation, where the assertion is not modified.
- **Delegation** goes beyond the forwarding scenario by adding information to the assertion that explicitly identifies all the parties through which a transaction flows.

Figure 4.3 depicts how delegation works using SAML. The User Agent (called the Client in the OAuth 2.0 model) gains access to Service Provider A (SP A), using its credentials stored in the Identity Provider (IdP). The SP A also obtains a SAML Assertion (identifying itself to the IdP using its credentials); hence, it can access the resource on SP B. The SAML Assertion of SP A is different respect to the User Agent version; it is modified. In the Figure, all steps characterizing the protocol are shown using the $A - D$ letters.

4.3. Comparison of OAuth 2.0 with SAML 2.0 Condition to Delegate. "OAuth 2.0 Threat Model and Security Considerations" [17] details the main threats and attacks on OAuth 2.0 as well as countermeasures to overcome them. One reason for the weakness of OAuth 2.0 is its evolution from OAuth 1.0 along with the adoption of incremental improvements that have exposed the system to possible flaws, such as:

a) compromising the communication among the parties;
b) obtaining client secrets; and
c) eavesdropping on access tokens.

The OAuth 2.0 protocol provides a greater degree of flexibility with respect to OAuth 1.0, especially in the way it can be applied and the use cases that it addresses, which may come at the price of security [17]. SAML is a much more mature framework conceived for many security purposes, in which the exchange of XML "assertion" guarantees a high degree of security. Especially if we consider the possibility offered by SAML to sign all communications with X509 certificates embedded into XML tags (see the XML Signature and XML Encryption Native Support of SAML 2.0 [23]). For example, XML Signature and XML Encryption help to avoid threats like *a*) and *b*) respectively. Given the greater security of the SAML, we chose it for our work, and in the next sections we describe our adoption of SAML for delegation, presenting detailed examples of SAML Assertions. Note also, that with the goal of improving the current OAuth 2.0 solution and reducing its weaknesses a working group of the IETF has recently released the draft: *SAML 2.0 Profile is used for OAuth 2.0 Client Authentication and Authorization Grants* [25], which describes a hybrid approach adopting SAML to strengthen OAuth 2.0.

5. VISION Cloud delegation scenarios. VISION Cloud addresses various use cases from different industries such as telco, media, healthcare and enterprise applications. Notably, all of them require some sort of delegation. Below, we describe some of the scenarios.

- **Delegation for Federated Resources** VISION Cloud also addresses scenarios where one cloud can *rent* external storage resources from other, foreign, clouds. The cloud needing these resource is responsible for accessing data objects in the new cloud on the behalf of the end user. The Delegation mechanism allows solving the access control problem of this scenario, where, as described above each cloud may have its own IAM.
- **Delegation for Storlets** VISION Cloud uses computational elements (special Agents called Storlets) for performing on-demand computations close to the data physical location. Access delegation is required to properly control a storlet's access to a user's resources on his behalf (e.g., storlets that perform data analytics or statistics should have the proper permissions from the resource owners).
- **Delegation for On-Boarding** On-boarding is a process in which a cloud customer migrates his data from one storage cloud (a New Cloud, i.e., Cloud A) to another cloud (an Old Cloud, i.e., Cloud B). Delegation mechanisms allow to grant the on-boarding component (termed the *Federator*) sufficient, but limited access rights to the customer resources.

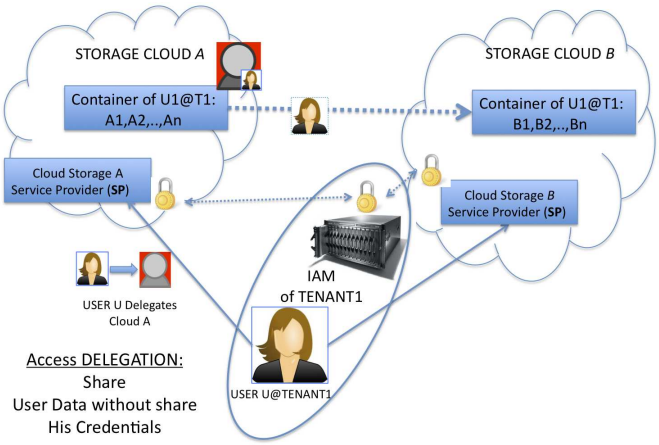


FIG. 5.1. *Delegation for VISION Cloud in On-Boarding Scenario*

In the remainder of the paper we use this last use case to illustrate delegation for on-boarding federation. Below we present the on-boarding federation system (see Figure 5.1). It was implemented as part of the VISION Cloud, but can also be easily added to other cloud systems wishing to provide a service for migration (on-boarding) from other providers. The new provider (Cloud A) is responsible for moving the customer data to itself from the old cloud provider (Cloud B). Furthermore, following the federation set up, applications and users begin accessing their data through the new cloud, which provides instant access to customer data remaining in the old cloud. Thus, the applications that access the migrated data are not influenced by on-boarding and can work transparently. The old cloud is assumed to be unaware of the on-boarding and is not required to introduce any modification. *Migrating data via on-boarding federation directly between the clouds leads to a significant savings in time and cost [30].* From the technical standpoint, the on-boarding architecture specifies the following three primary flows:

1. *On-boarding set-up:* An on-boarding relationship between a container in the old cloud and a container in the new cloud is set up and persisted through the protected metadata of the container on the new cloud.
2. *Direct access:* Once the relationship is set-up, all client's applications may start to immediately access the objects of the old container through the new cloud. When a client accesses an object on the new cloud that has not yet been on-boarded, the Federator module gets the object from the old cloud and puts it in the new cloud on the behalf of the end user.

3. *Background on-boarding:* The Federator component creates background jobs on the the cloud that fetch objects from the container in the old cloud and copy them to the container in the new cloud. These jobs run when the resource utilization (e.g., CPU and network) in the new cloud is low so that they do not interfere with the normal operation of the cloud. These jobs also need authorization from the user to access the old cloud and depending on the architecture may also need authorization the write the objects on the new cloud.

The delegation architecture to provide the authorization required for the direct and background on-boarding is described in the next section.

6. VISION Cloud on-boarding delegation architecture. We first describe the flow for the SAML-based delegation mechanism that we have chosen, and then show how we apply it for on-boarding.

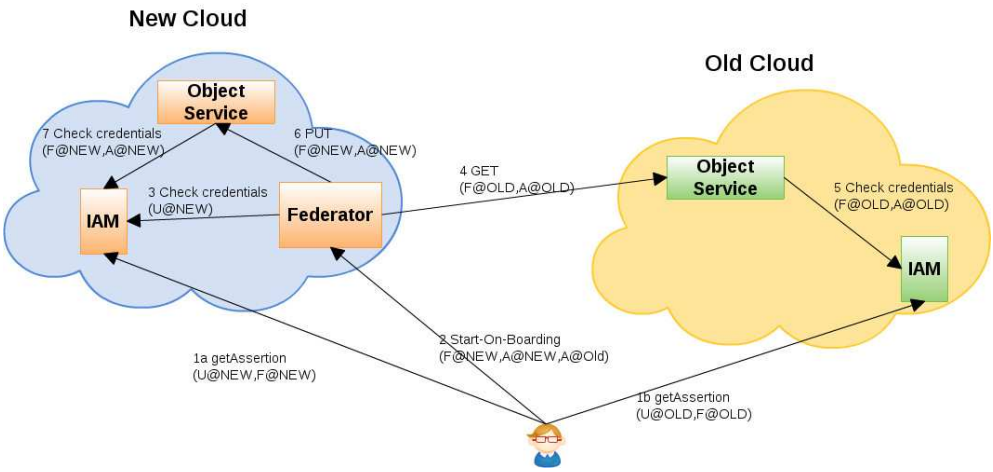


FIG. 6.1. *Delegation for on-boarding in VISION Cloud*

Based on the comparison presented in Sect. 4.1 we chose to implement the delegation using SAML2Del. In this protocol, the delegation document is a signed SAML assertion containing the details of the delegation, held in a specific field called *Condition*. To describe the delegation flow, we consider a simple setup with single IAM and two accounts, U1 and U2, which correspond to the user in the old and new clouds respectively, where a user can also be a process or an infrastructure.

The access flow is as follows:

1. U1 logs into the IAM using her credentials and asks to generate a signed delegation assertion stating that U2 is authorized to perform, for example, GET operations on behalf of U1 for one day.
2. U1 provides the assertion to U2.
3. U2 logs into the IAM using her credentials, provides the assertion and is authorized to perform GET operations for the entire day.

In the on-boarding federation scenario, the Federator reads the objects on behalf of the user U1@Old in the Old Cloud, and writes them on the behalf of the user U1@New in the New Cloud, where @Old and @New correspond to the different identity management servers of the two clouds.

6.1. On-boarding Delegation Flow. Figure 6.1 shows the on-boarding delegation flow between two VISION Clouds: the New and the Old, where each cloud has its own IAM. Each IAM incorporates an Identity Provider (IdP) and a Service Provider (SP) in order to carry out the SAML protocol. The figure shows a Federator component which can be introduced either independently or as an added-value feature of the New Cloud. Each cloud also has an Object Service, which carries out the basic data access operations such as GET/PUT of objects and containers.

The Federator has an identity in both clouds. In particular, let $F@OLD$, denote the federator identity (F) in the IAM of the Old Cloud and let $F@NEW$ denote its identity (F) in the IAM of the New Cloud. The user (U) that requests the on-boarding also has an identity on each cloud, in particular, $U@OLD$ (in the Old

Cloud) and $U@NEW$ (on the New Cloud). All the identities must be registered with suitable credentials (e.g. username/password) in their corresponding IAMs.

The delegation flow is modeled using two delegations, in particular: (1) $U@OLD$ delegates $F@OLD$ to GET her objects from her container on Old Cloud; and (2) $U@NEW$ delegates $F@NEW$ to PUT objects on her container in New Cloud.

The following operations are performed during the delegation flow for on-boarding.

1. The user logs in and obtains delegation assertions for both clouds:
   (a) Delegation assertion ($A@NEW$), based on the credentials of $U@NEW$ and the userid of $F@NEW$.
   (b) Delegation assertion ($A@OLD$), based on the credentials of $U@OLD$ and the userid of $F@OLD$.
2. The user starts the federation process, providing the following parameters:
   (a) $U@NEW$ - credentials to activate the Federator.
   (b) $A@NEW$ - the assertion ID delegating $F@NEW$ to ask for PUT operations on behalf of $U@NEW$.
   (c) $A@OLD$ - the assertion ID delegating $F@OLD$ to ask for GET operations on behalf of $U@OLD$.
3. The Federator authenticates $U@NEW$ through the IAM of the New Cloud.
4. The Federator performs a GET operation to the Old Cloud using $F@OLD$ credentials as delegate of $U@OLD$.
5. The Object Service of the Old Cloud checks the credentials and the delegation with the IAM of the Old Cloud.
6. The Federator PUTs the retrieved object to the New Cloud using $F@NEW$ credentials as Delegate of $U@NEW$.
7. The Object Service of the New Cloud checks the credentials and the delegation with the IAM of the New Cloud.

6.2. Delegation Advantages. The Federator acts as a canonical user, authenticating itself for each REST request (GET from Old Cloud and PUT to New Cloud). The delegation protocol allows it to access the user resources without possessing the user credentials or requiring an explicit trust relationship with the Old Cloud. The assertion IDs are inserted into the REST messages and they are cached by the IAM for increasing the performances (as below). In addition to preserving the security, this scheme has the advantage of an easy combination with other authorization methods used in the underlying object store system. For example, VISION Cloud uses Access Control Lists (ACLs), which are associated with every resource, such as object or container. ACLs are configured with a RESTful API as specified by CDMI, where the syntax is NFSv4 compatible allowing compatibility with traditional file systems. Since ACLs are distributed with the resource, they provide an efficient way of setting fine-grained access control with per object/container permissions. Notably, by granting the Federator specific but limited permissions to act on the behalf of the end user, the described delegation allows to properly preserve all of the permissions set by ACLs. Specifically, after the Object Service in Figure 6.1 checks the Federator credentials and assertion (Step 7), the system drops the privileges to those defined in the assertion (the identity and the delegated role granted to the Federator by the end user). Due to the mechanism of role delegation the Federator can securely access the resources without the need to specifically add it to the ACLs.

7. Related Work. Below, we review the models for federation between cloud operators and then focus on works for federation of access control for identity federation and delegation.

Usually federation consists of the establishment of a trust context between parties with the purpose of benefiting of business advantages. According to the view of the VISION Cloud project, Cloud federation may represent a compelling business model for SMEs, where many stakeholders (i.e., Cloud providers, tenants and customers) interact with each other for creating new opportunities and satisfying even more needs [12]. There are four basic capabilities that characterize each entity of the federation, which "keeps authority about the information passed to the other entities of the federation" and "has authority to create a global view of the data that is available among all the entities of the federation". An entity of the federation "is not forced to perform tasks of another entity of the federation" and "can autonomously decide to enter or leave the federation". Examples of such systems were presented by Tordsson et al. [29], Kurze et al. [14] as well as the recent Colony system, that federated several Openstack Swift deployments in a Colony by introducing a Swift Dispatcher component, prefixing the container names. Unlike this work, we do not assume any naming conventions or

architectural components common to the two clouds. The federation described in [6] is aimed at computation management (consolidation of VMs among Clouds). It shows similarities to the work done by Celesti et al. [3] and Rochwerger et al. [21], where the federation problems for data management between several IaaSs are addressed. In the latter case the authors described how to elastically enlarge in a transparent way, the physical resource of cooperating IaaSs.

From the security and privacy standpoint, the cloud has not kept pace with the enormous volumes of user identities that network administrators must manage and secure. An identity fabric that links multiple applications to a single identity would address this problem, enabling full-scale cloud adoption as it is highlighted in the *Architecting a Cloud-Scale Identity Fabric* (see [18]). The authors in [4] identified the services and APIs are necessary to be realized for accessing cloud resources in a useful and focused way. They range from: *Federated Identity*, to *Delegation Of Authority*, and *Levels Of Assurance*, *Attributes*, *Access Rights*, till *Authorization*. In this work, the authors described all APIs they introduced, and applied them to a real cloud middleware, the Eucalyptus S3 Service. This work explains how the researchers have used these proof-of-concept APIs and how to exploit and match them with the existing services of Kent University.

The Security Assertion Markup Language (SAML) became a popular technology for solving issues related to the federation needs. Indeed many existing works provide solutions for a variety of scenarios that are based on the SAML mechanism and its capabilities. These solutions do not solve all the cloud federation problems. The federation problem in cloud computing is greater than the one in traditional systems.

The first practice of cooperation between providers was about sharing the network traffic. A few years later service federation over the Internet has become a well established approach: it had to be supported by a mechanism for trusting identities across different domains, which is identity federation. The latest trend to federate identities over the Internet is represented by the Identity Provider/Service Provider (IdP/SP) model [16], supported by digital certificates.

An important example of a popular implementation of SAML is Shibboleth [26]. Unfortunately, most systems still lack important capabilities required for the federation among different cloud providers. The main constraint of the existing federation solutions is that they are designed for static environments requiring a priori policy agreements, whereas clouds are dynamic and heterogeneous environments which require particular security and policy arrangements.

Interoperability in federated heterogeneous cloud environments is addressed in [15], in which the authors propose a trust model where the trust is delegated between trustworthy parties satisfying certain constrains. The approach in VISION Cloud considers also the issue with the data location and protection with fine-grain ACLs. Pearson at al. [19] also introduce a privacy manager, taking care of data compliance according to the laws of different jurisdictions.

Huang et al. present [9] an Identity Federation Broker for service clouds. They address federation in the context of SaaS services in which SAML represents the basic protocol. They make use of a trusted third party as a trust broker to simplify the management of identity federation in a user centric way.

A recent work called *FACIUS* describes the use of SAML for Non Web-Based Services [11]. The authors make an assessment of a system accomplishing a real implementation of SAML with SSH. In particular they have reported evaluations with respect to requirements, performance, security, and legal aspects.

The works presented above show various SAML applications and justify its adoption in our solution. However, the they consider it in a different aspect which is not suitable for our aims. Here, in addition to presenting standard SAML flows, we discuss a slightly modified version allowing to simplify the on-boarding federation.

When considering the delegation technologies, protocols leveraging the Public-Key Infrastructures (PKI) was widely adopted grid systems. For example, grids make use of signed certificates for users' identification and jobs submission. In grid terminology, Virtual Organizations (VOs) represent entities that are able to manage certificates, distributing and verifying them, accomplishing a full PKI. Right now, a recent trend for managing credential and delegations in federated cloud scenarios appears to make use of X.509 certificates.

In the direction of managing identity and authorization for Community Clouds using PKI is discussed in [5]. The authors introduce an *identity broker* to bind the Web Single Sign-on to a key-based system. In particular they implemented a solution (*libabac* package) using the Attribute-based Access Control (ABAC) and the role-based trust management (RT): RT/ABAC. The *libabac* uses X.509 as a transport. RT/ABAC credentials are

X.509 attribute certificates. The RT/ABAC may handle the processing of delegation chains.

A recent and detailed work on delegation is described in the paper entitled: *OAuth and ABE based Authorization in Semi-Trusted Cloud computing, [28].* Here, the authors enhanced the OAuth capabilities using the encryption in attribute-based access control system exploiting metadata. They introduced a complex model in which authentication, authorization, delegation, access trees, new tokens, time slots, user certificates are investigated. Their objective is to provide a complete solution with a high level of flexibility useful in many cloud scenarios.

In spite this progress early storage cloud solutions focused on scalability and had difficulties in achieving the delegation requirements [8]. However, in recent years they are slowly adopting more advanced access control technologies. For example, the OpenStack security and access control component Keystone has recently adopted PKI (see [10]). Keystone uses tokens, which are json documents that contain the required access control information about users, projects, roles and domains. Starting from the Grizzly release of Openstack, these tokens are cryptographically signed based on the X509 standard. Keystone serves as a Certificate Authority (CA) and uses its signing key and certificate to sign tokens. Thanks to PKI, other OpenStack entities can then locally verify these tokens with the public key without the need to contact Keystone.

8. Conclusions and Future Work. In this paper we presented the concept of delegation for on-boarding federation between storage clouds. Our solution is based on the SAML 2.0 Condition to Delegate extension. The added value of this work is in considering all security implications in using the delegation technique for the on-boarding procedure. We also show how this delegation architecture can be implemented using SAML solution. We implemented the solution as part of our work on VISION Cloud, accomplishing RESTful APIs and creating new SAML delegation Assertions. In the future we will evaluate the impact of it on VISION Cloud architecture, analyzing its complexity and performance.

Acknowledgments. The research leading to the results presented in this paper has received funding from the European Union's Seventh Framework Programme (FP7 2007-2013) Project VISION-Cloud under grant agreement number 217019.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia, *Above the Clouds: A Berkeley View of Cloud Computing*, Tech. Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[2] CAS, *Central authentication service,*. http://www.jasig.org/cas, June 2013.

[3] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, *Integration of clever clouds with third party software systems through a rest web service interface*, in Proceedings - IEEE Symposium on Computers and Communications, 2012, pp. 827–832.

[4] D. W. Chadwick and M. Casenove, *Security apis for my private cloud - granting access to anyone, from anywhere at any time*, in Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, CLOUDCOM '11, Washington, DC, USA, 2011, IEEE Computer Society, pp. 792–798.

[5] J. Chase and P. Jaipuria, *Managing identity and authorization for community clouds*, tech. report, Department of Computer Science, Duke University, 2012. Technical Report CS-2012-08.

[6] I. Goiri, J. Guitart, and J. Torres, *Characterizing cloud federation for enhancing providers' profit*, in Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, Jul 2010, pp. 123–130.

[7] D. Hardt, *The OAuth 2.0 Authorization Framework*. RFC 6749 (Proposed Standard), Oct. 2012.

[8] D. Harnik, E. K. Kolodner, S. Ronen, J. Satran, A. Shulman-Peleg, and S. Tal, *Secure access mechanism for cloud storage*, Scalable Computing: Practice and Experience, 12 (2011).

[9] H. Y. Huang, B. Wang, X. X. Liu, and J. M. Xu, *Identity federation broker for service cloud*, in Service Sciences (ICSS), 2010 International Conference on, May 2010, pp. 115–120.

[10] KEYSTONE, *Welcome to keystone, the openstack identity service*. http://docs.openstack.org/developer/keystone, 2013.

[11] J. Kohler, S. Labitzke, M. Simon, M. Nussbaumer, and H. Hartenstein, *Facius: An easy-to-deploy saml-based approach to federate non web-based services*, in Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on, Jun 2012, pp. 557–564.

[12] E. K. Kolodner, A. Shulman-Peleg, D. Naor, P. Brand, M. Dao, A. Eckert, S. Gogouvitis, D. Harnik, M. Jaeger, D. Kyriazis, et al., *Data intensive storage services on clouds: Limitations, challenges and enablers*, European Research Activities in Cloud Computing, D. Petcu and JL Vazquez-Poletti, Eds. Cambridge Scholars Publishing, (2012), pp. 68–96.

[13] E. K. Kolodner, S. Tal, D. Kyriazis, D. Naor, M. Allalouf, L. Bonelli, P. Brand, A. Eckert, E. Elmroth, S. V. Gogouvitis, D. Harnik, F. Hernández, M. C. Jaeger, E. B. Lakew, J. M. Lopez, M. Lorenz, A. Messina, A. Shulman-Peleg, R. Talyansky, A. Voulodimos, and Y. Wolfsthal, *A cloud environment for data-intensive storage services*, in CloudCom, 2011, pp. 357–366.

[14] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze, *Cloud federation*, in Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011), IARIA, Sep 2011.

[15] W. Li and L. Ping, *Trust model to enhance security and interoperability of cloud environment*, in Cloud Computing, Nov 2009, pp. 69–79.

[16] Liberty, *An alliance project*. http://projectliberty.org, 2013.

[17] T. Lodderstedt, M. McGloin, and P. Hunt, *OAuth 2.0 Threat Model and Security Considerations*. RFC 6819 (Informational), Jan. 2013.

[18] E. Olden, *Architecting a cloud-scale identity fabric*, Computer, 44 (2011), pp. 52–59.

[19] S. Pearson, Y. Shen, and M. Mowbray, *A privacy manager for cloud computing*, in Cloud Computing, Nov 2009, pp. 90–106.

[20] D. Recordon and D. Reed, *Openid 2.0: a platform for user-centric identity management*, in Proceedings of the second ACM workshop on Digital identity management, DIM '06, New York, NY, USA, 2006, ACM, pp. 11–16.

[21] B. Rochwerger, S. Naqvi, C. Ponsard, J. Latanicki, P. Massonet, and M. Villari, *A monitoring and audit logging architecture for data location compliance in federated cloud infrastructures*, in IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, 2011, pp. 1510–1517.

[22] SAML-DEL, *V2.0 condition for delegation*. http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-delegation-cs-01.pdf, 2013.

[23] SAML-ENHANC, *2.0 enhancements*. http://saml.xml.org/saml-2-0-enhancements, 2007.

[24] SAML-OASIS, *V2.0 technical*. http://www.oasis-open.org/specs/index.php, Jan 2013.

[25] SAML-OAUTH, *Saml 2.0 profile for oauth 2.0 client authentication and authorization grants", note="http://datatracker.ietf.org/doc/draft-ietf-oauth-saml2-bearer, 2013.*

[26] Shibboleth, *System standards*. http://shibboleth.internet2.edu/, Jan 2012.

[27] SWIFT, *Welcome to Swift's documentation*, June, 2013. http://http://docs.openstack.org/developer/swift/.

[28] A. Tassanaviboon and G. Gong, *Oauth and abe based authorization in semi-trusted cloud computing: aauth*, in Proceedings of the second international workshop on Data intensive computing in the clouds, DataCloud-SC '11, New York, NY, USA, 2011, ACM, pp. 41–50.

[29] J. Tordssona, R. S. Monterob, R. Moreno-Vozmedianob, and I. M. Llorente, *Optimized placement of a computational cluster across multiple clouds*.

[30] G. Vernik, A. Shulman-Peleg, S. Dippl, C. Formisano, M. Jaeger, E. Kolodner, and M. Villari, *Data on-boarding in federated storage clouds*, in IEEE CLOUD 2013 IEEE 6th International Conference on Cloud Computing June 27-July 2, 2013, Santa Clara Marriott, CA, USA (Center of Silicon Valley), 2013.

[31] XACML, *Cross-enterprise security and privacy authorization (xspa) profile of xacmlv2.0 for healthcare version 1.0*. http://www.oasis-open.org/committees/document.php?document_id=34164&wg_abbrev=xacml.

Appendix A. On-Boarding Delegation details: REST API.

In this section we present the API of VISION Cloud for delegation, first a description of the parameters and then some details of the RESTful API for creating and using a delegation.

A.1. API parameters. Table A.1 shows the parameters to be provided by an end-user (the delegator) during the interaction with his IAM to create a delegation document and the delegation token representing it to be granted to the delegate. The delegator provides his credentials, and describes the capabilities and the actions he wishes to delegate, including a limitation of the delegation to certain containers, and identifies the delegate.

Name	Type	Mandatory/Optional	Description
delegatorUsername	JSON String	mandatory	Name of the delegator user.
delegatorPassword	JSON String	mandatory	Password of the user.
delegatorTenant	JSON String	mandatory	Tenant of the user.
delegatedId	JSON String	mandatory	Identifier of the delegated user.
delegatedTenant	JSON String	mandatory	Name of the delegated tenant.
delegatedRoles	JSON Array	mandatory	Roles of the delegator (to be delegated).
delegatedActions	JSON Array	mandatory	Actions of the delegator (to be delegated).
delegatedContainer	JSON String	mandatory	Name of the container (to be delegated)

Table A.1
Delegation capabilities provided to a delegated user and/or software component *(e.g., the Federator).*

A.2. RESTful API for on-boarding delegation. In VISION Cloud storage related operations (create/read/update/delete) are invoked through a RESTful API over standard HTTP. A token for a delegation assertion is also obtained through a RESTful request. In particular, Listing 1 shows a request to create a delegation assertion and return a token for it, e.g., as required in steps 1a and 1b of Figure 6.1.

Listing 1
HTTP - User Delegation Request

```
Request:
POST <root-uri>/visionIAM/requestDelegation
        Accept: application/json
        Content-Type: application/json
        {
                "delegatorUsername":"delegator_username",
                "delegatorPassword":"delegator_password",
                "delegatorTenant":"delegator_tenant",
                "delegatedId":"delegated_identifier",
                "delegatedTenant":"delegated_tenant",
                "delegatedRoles":["role1","role2",...,"roleN"],
                "delegatedActions":["action1","action2",...,"actionN"],
                "delegatedContainer":"delegated_container"
        }
```

If the delegator is authorized to perform a delegation for the required roles and operations, the response is a 200 OK and contains the the delegation token. Such a response is shown in Listing 2. If the delegator is not authorized then the response is a 400.

Listing 2
HTTP - IAM Delegation Token Released

```
Response:
HTTP Status
```

```
HTTP/1.1 200OK
Content-Type: application/json
{
"delegationToken":"NGFiYWVmOTctMmY0MS00MTgyLWFkYTItODc0M2EyMDA1MDZi"
}
```

A RESTful request on VISION Cloud, e.g., a GET request to read an object, contains an authorization header holding the credentials (username/password) for the user making the request. The VISION Cloud Identity Management System uses these credentials to authenticate the user. The same header is used for a request on VISION Cloud from a delegate; in this case the header contains the credentials of the delegate and the delegation token. Listing 3 shows an authorization header containing a delegation token.

Listing 3
HTTP - Delegated Access

```
Request:
GET <root-uri>/visionIAM/authenticate HTTP/1.1
Authorization: DEL base64Encoded{username@tenantName:password:delegationToken}
```

In SAML all messages are signed with X509 certificates, to avoid any kind of attack when managing identities, roles and actions (Authentication and Authorization phases). The signature is done with valid (recognized) certificates and thus is not exploitable for an attack. Section. B briefly describes these SAML security capabilities.

Appendix B. SAML Assertion for On-Boarding Delegation.

Fig. B.1. *SAML Assertion used for the VISION Cloud On-Boarding Delegation scenario*

This section highlights the various parts comprising a SAML Assertion used for the on-boarding delegation scenario. As can be seen, the assertion is an XML-based container of information. In general all SAML assertions are associated with a subject (< *Subject* > element, see Listing 6. The standard defines three statements: *Authentication Assertion*, *Attribute Assertion* (see Listing 7), and an *Authorization Decision Assertion*. In SAML there are also *Signatures*, *Conditions* (see Listing 6) and *Issuers*. The prefix "saml" represents the basic namespace for a SAML V2.0 assertion (see Listing 4). In our case the SAML Assertion has 6 parts as depicted in Figure B.1. In all listings reported below we highlight the flexibility of the SAML standard as well as its strong security.

Listing 4
SAML - Root container and Issuer

```
<?xml version="1.0" encoding="UTF-8"?>
<saml2:Assertion
    xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
    ID="_9a5a551aeb58718ed0076e31d1cf510b" IssueInstant="2013-10-29T10:42:25.908Z"
    Version="2.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
......
</saml2:Assertion>
------
  <saml2:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
  https://identityProvider.hostname/idp
  </saml2:Issuer>
------
```

Listing 5
SAML - Signature parts

```
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
              <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"
                    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"/>
              <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"
                    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"/>
              <ds:Reference URI="#_8f1c03bb-bcb1-48eb-8a52-33c19e8a5d66"
              xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:Transforms xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                   <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature
                       " xmlns:ds="http://www.w3.org/2000/09/xmldsig#"/>
                   <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" xmlns:ds="
                       http://www.w3.org/2000/09/xmldsig#">
                     <ec:InclusiveNamespaces PrefixList="del xs" xmlns:ec="http://www.w3.org
                         /2001/10/xml-exc-c14n#"/>
                   </ds:Transform>
                </ds:Transforms>
                <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" xmlns:ds="
                     http://www.w3.org/2000/09/xmldsig#"/>
                <ds:DigestValue xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...
                </ds:DigestValue>
              </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...
            </ds:SignatureValue>
            <ds:KeyInfo>
              <ds:X509Data>
                <ds:X509Certificate>...</ds:X509Certificate>
              </ds:X509Data>
            </ds:KeyInfo>
      </ds:Signature>
```

Listing 6
SAML - Subject and Conditions

```
1
2  <saml2:Subject>
3        <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
4        NameQualifier="https://identityProvider.hostname/idp"
5        SPNameQualifier="https://serviceProvider.hostname/sp">
6        _a36102490a9b9c196255ff86ee1a052f</saml2:NameID>
7            <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
8               <saml2:SubjectConfirmationData>
9                 <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
10                  <ds:X509Data>
11                    <ds:X509Certificate>...</ds:X509Certificate>
12                    <ds:X509SubjectName>...</ds:X509SubjectName>
13                    <ds:X509IssuerSerial>
14                      <ds:X509IssuerName>...</ds:X509IssuerName>
15                      <ds:X509SerialNumber>...</ds:X509SerialNumber>
16                    </ds:X509IssuerSerial>
17                  </ds:X509Data>
18                </ds:KeyInfo>
19              </saml2:SubjectConfirmationData>
20             </saml2:SubjectConfirmation>
21    </saml2:Subject>
22 ---------
23    <saml2:Conditions>
24        <saml2:Condition
25                        NotBefore="2013-10-31T09:40:00.440Z"
26               NotOnOrAfter="2013-10-31T18:40:00.440Z"
27               xmlns:del="urn:oasis:names:tc:SAML:2.0:conditions:delegation"
28               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
29               xsi:type="del:DelegationRestrictionType">
30               <del:Delegate ConfirmationMethod="#sender-vouches"
31               DelegationInstant="2013-10-29T10:42:25.900Z">
32               <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
33               delegated</saml2:NameID>
34           </del:Delegate>
35        </saml2:Condition>
36    </saml2:Conditions>
```

Listing 6 shows the flexibility of SAML in defining *Conditions*. In this case the Assertion is valid for a specific time interval as highlighted in NotBefore and NotOnOrAfter Condition Attributes (see lines 25 and 26). A timestamp allows tracking the creation time of a delegation (see line 31). The *Subject* tag describes all the fields of a X509 user/host certificate necessary for ensuring security (see from line 9 to 18).

Listing 7
SAML - Authentication and Attribute Statements

```
1    <saml2:AuthnStatement AuthnInstant="2011-03-29T16:35:12.440Z" SessionIndex="123456">
2        <saml2:AuthnContext>
3               <saml2:AuthnContextClassRef>
4                             urn:oasis:names:tc:SAML:2.0:ac:classes:PreviousSession
5               </saml2:AuthnContextClassRef>
6        </saml2:AuthnContext>
7    </saml2:AuthnStatement>
8   -------
9    <saml2:AttributeStatement>
10       <saml2:Attribute FriendlyName="delegated_roles" Name="delegated_roles">
11           <saml2:AttributeValue
12               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
13               xsi:type="xs:string">REMOTE_ADVISOR
14           </saml2:AttributeValue>
15       </saml2:Attribute>
16       <saml2:Attribute FriendlyName="delegated_actions" Name="delegated_actions">
17           <saml2:AttributeValue
18               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
19                xsi:type="xs:string">CREATE_OBJECT
20            </saml2:AttributeValue>
21            <saml2:AttributeValue
22                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
23                xsi:type="xs:string">DELETE_OBJECT
24            </saml2:AttributeValue>
25        </saml2:Attribute>
26        <saml2:Attribute FriendlyName="delegated_container"
27        Name="delegated_container">
28            <saml2:AttributeValue
29                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
30                xsi:type="xs:string">7D1FG1
31            </saml2:AttributeValue>
32        </saml2:Attribute>
33        <saml2:Attribute FriendlyName="delegator_username"
34        Name="delegator_username">
35            <saml2:AttributeValue
36                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
37                xsi:type="xs:string">delegator
38            </saml2:AttributeValue>
39        </saml2:Attribute>
40        <saml2:Attribute FriendlyName="delegator_username"
41        Name="delegator_username">
42            <saml2:AttributeValue
43                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
44                xsi:type="xs:string">delegationTenant
45            </saml2:AttributeValue>
46        </saml2:Attribute>
47        <saml2:Attribute FriendlyName="delegated_username"
48        Name="delegated_username">
49            <saml2:AttributeValue
50                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
51                xsi:type="xs:string">delegated
52            </saml2:AttributeValue>
53        </saml2:Attribute>
54        <saml2:Attribute FriendlyName="delegated_tenant"
55        Name="delegated_tenant">
56            <saml2:AttributeValue
57                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
58                xsi:type="xs:string">delegationTenant
59            </saml2:AttributeValue>
60        </saml2:Attribute>
61    </saml2:AttributeStatement>
```

As can seen in Listings 5 and 6, 50% of the SAML structure is devoted to the security definitions (certificates X509, keys, sec algorithm, digests, etc.). These parts are necessary for securing any complex distributed scenario where many actors are involved. The complexity increases if we consider the case of federation among clouds. We remark that other solutions for delegation do not have the same level of accuracy and security in treating and exchanging sensitive data as does SAML. All Actions, Roles and Resources introduced in our implementation are described in the $< saml2 : AttributeStatement >$ statement (see Listing 7). In particular, all fields of table A.1 are reported in the listing from the line 9 to 61.

Edited by: Maria Fazio and Nik Bessis
Received: Nov 2, 2013
Accepted: Jan 10, 2014

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2_ε using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.