Future Internet **2014**, *6*, 126-154; doi:10.3390/fi6010126

OPEN ACCESS

future internet

ISSN 1999-5903

www.mdpi.com/journal/futureinternet

Article

Information Integration Platform for Patient-Centric Healthcare Services: Design, Prototype and Dependability Aspects

Yohanes Baptista Dafferianto Trinugroho

Department of Information and Communication Technology, University of Agder, Jon Lilletuns vei 9, Grimstad 4879, Norway; E-Mail: dafferianto.trinugroho@uia.no; Tel.: +47-3723-3791

Received: 1 November 2013; in revised form: 30 January 2014 / Accepted: 8 February 2014 / Published: 6 March 2014

Abstract: Technology innovations have pushed today's healthcare sector to an unprecedented new level. Various portable and wearable medical and fitness devices are being sold in the consumer market to provide the self-empowerment of a healthier lifestyle to society. Many vendors provide additional cloud-based services for devices they manufacture, enabling the users to visualize, store and share the gathered information through the Internet. However, most of these services are integrated with the devices in a closed "silo" manner, where the devices can only be used with the provided services. To tackle this issue, an information integration platform (IIP) has been developed to support communications between devices and Internet-based services in an event-driven fashion by adopting service-oriented architecture (SOA) principles and a publish/subscribe messaging pattern. It follows the "Internet of Things" (IoT) idea of connecting everyday objects to various networks and to enable the dissemination of the gathered information to the global information space through the Internet. A patient-centric healthcare service environment is chosen as the target scenario for the deployment of the platform, as this is a domain where IoT can have a direct positive impact on quality of life enhancement. This paper describes the developed platform, with emphasis on dependability aspects, including availability, scalability and security.

Keywords: Internet of Things; integration platform; middleware; REST; publish/subscribe; availability; scalability; security; healthcare

1. Introduction

The term "Internet of Things" (IoT) was popularized at the Massachusetts Institute of Technology (MIT) Auto-ID Center in 1999, where a group of people started to design and propagate a cross-company radio-frequency identification (RFID) infrastructure [1,2]. Advancements in information and communications technology (ICT) have enabled IoT's vision to be realized by turning everyday objects into connected objects [3,4], so that the information gathered (or "sensed") by these objects can be used in various different services. Everyday objects can be employed to capture and create information from the physical world instead of relying purely on people, as normally done in traditional information systems [1]. This is mainly achieved by using RFID and sensor technologies. The ability to react to events in the physical world automatically not only opens up new opportunities for dealing with complex or critical situations, but also enables a wide variety of business processes to be optimized. The real-time interpretation of data from the physical world can lead to the introduction of various novel services and may deliver substantial economic and social benefits [4].

Although localized services, utilizing information gathered from nearby objects, can be useful in many different scenarios, the global pervasiveness of things can only be realized when these everyday objects are connected to the Internet [5]. The Internet has become the *de facto* standard backbone for deploying services beyond time and space barriers, which enables people or other services to consume them 24/7 from all over the world. Cloud computing has pushed the boundary even further, enabling developers with innovative ideas to develop and deploy services without large capital outlays in hardware [6]. Myriad Internet-based services can make use of the collected information from various different everyday objects for value-added functionalities. However, everyday objects may have limitations in terms of processing power or battery lifetime, which makes it infeasible to incorporate a full transmission control protocol/internet protocol (TCP/IP) stack [7] in order to communicate with the current Internet infrastructure. Other more power-preserving communications protocol stacks (e.g., Bluetooth, ZigBee, ANT) are more commonly used in embedded devices. In order to be connected to the Internet, additional gateway devices that have implemented full TCP/IP stacks are needed. These gateway devices should have at least two network interfaces, one facing the connected objects and another one facing the Internet, and physically can range from dedicated servers to mobile devices (e.g., smartphones, tablets). The latter is particularly useful for scenarios that involve the mobility of the users [8]. With the increasing needs of everyday objects to be connected to the Internet, new technologies have been developed to extend the Internet to small devices [9,10], such as Internet protocol version 6 (IPv6) over low-power area networks (6LoWPAN) [11,12] and GLoWBALIPv6 [13].

When everyday objects are connected to the Internet (e.g., through a smartphone gateway), an application-layer protocol is needed to communicate with Internet-based services that are interested in using the gathered information. Within the web domain, hypertext transfer protocol (HTTP) [14] has been widely used to exchange content over the Internet [15] since the inception of the World Wide Web (WWW) [16]. Web services, which are software systems designed to support interoperable machine-to-machine interaction over a network, normally use HTTP to convey messages, as well. This is true for both traditional web services [17] and RESTful web services [18]. The majority of Internet-based services provide web service interfaces to enable message exchange with external systems. Thus, the

HTTP protocol, combined with the web service approach, is a good combination to be used for message exchange in the application layer.

The IoT plays an important role in healthcare, for example, in general remote vital sign monitoring of patients [19], as well as in specific chronic disease treatment, such as diabetes therapy management [20]. Within the personal healthcare sector, many portable and wearable medical and fitness devices are being pushed to the consumer market by various vendors. This can be seen as a positive trend towards the self-empowerment of a healthier lifestyle and enables healthcare workers to more efficiently keep track of their patients' health conditions by means of telecare, if such a feature is provided. Many vendors provide additional online services for devices they sell, enabling users to better visualize, store and share the gathered information from the devices through the Internet. However, many of these services are integrated with the devices following a closed vertical "silo" approach [21], where the devices can only be used with the provided services, and different services from other vendors cannot make use of the gathered information. The main disadvantage of this situation is the inability to combine information gathered from different devices produced by different vendors for better reasoning and decision making [22,23]. To solve this issue, open interfaces (e.g., web service interfaces) have to be provided by device vendors, so that service developers can incorporate the information collected from the devices in their services.

A common way to integrate devices and Internet-based services is to directly exchange messages between the two parties in a point-to-point manner. The downside of this approach is that devices that "sense" new information should deliver it to all services that are interested in consuming it, either through a push approach from the devices or in a pull fashion from the services. This can be a major drawback from an energy efficiency standpoint, as portable and wearable connected devices commonly run on batteries, and thus, sending similar information to many destinations (*i.e.*, services) will lead to a shorter lifetime of the devices. With the proliferation of mobile cloud computing usage in recent years [24–26], a brokered approach with a service broker being deployed in the cloud can be a good alternative, since the devices need to connect to the Internet in order to communicate with Internet-based services anyway. The service broker will handle the message delivery tasks to all interested services, so that the devices only need to send the collected information once. A publish/subscribe messaging pattern is advantageous in such a broker, so that services interested in specific information can subscribe to that particular information and get notification from the broker whenever new information is available. However, such a service broker can be seen as a single point of failure, since various devices and services rely on it, and thus, its dependability is very crucial. An information integration platform (IIP), which acts as a service broker between connected devices and Internet-based services, has been proposed and developed as a prototype. Several services within the healthcare domain, mainly related to telecare, have been developed on top of the platform, as well. This paper will briefly describe the main functionalities of the platform, while focusing more on its dependability aspects. Prototype services on top of it will be described to put the platform into a deployment context within the healthcare domain.

2. Related Work

There are several existing works related to the proposed platform that have been conducted in the past. This section describes some of them briefly.

2.1. Publish/Subscribe Middlewares

The publish/subscribe messaging pattern was introduced more than a decade ago and is still considered to be one of the most important communications mechanisms, as it is well adapted to the loosely coupled nature of distributed interaction in large-scale applications. Subscribers have the ability to express their interest in an event and are subsequently notified of any event that is generated by a publisher and matches their registered interest [27]. This complies with event-driven architecture, where an event is asynchronously propagated to all subscribers.

Many existing middleware prototypes make use of the publish/subscribe messaging pattern, both in research and production arenas. In the research domain, for example, there are Gryphon [28], Hermes [29], java event-based distributed infrastructure (JEDI) [30], Scribe [31] and scalable internet event notification architectures (SIENA) [32]. Although these systems provide relatively complex topic-based, content-based or type-based subscription schemes, they mainly provide programming language-specific application programming interfaces (APIs), such as for Java or C++.

Within the web services arena, web services eventing (WS-Eventing) and web services notification (WS-Notification) are two major competing specifications for "big" web services to incorporate the publish/subscribe message exchange paradigm. Although their architectures differ from each other, convergence between the two was realized to a certain degree [33]. Web services messenger (WS-Messenger) [34] is a middleware prototype that aims to mediate WS-Eventing and WS-Notification specifications by enabling the utilization of existing messaging systems to provide scalable subscription management and message delivery.

However, there is no standard initiative for RESTful web services to include publish/subscribe for supporting an asynchronous communications model. Various projects proposed different approaches to add a publish/subscribe functionality to RESTful web services-based middleware. Universal presence service (UnivPS) was proposed in [35], which provides publish/subscribe functionality for a presence service by using REST interfaces in the telecommunications domain. The authors in [36] proposed a RESTful gateway for integrating smart meters in future houses that makes use of topic-based publish/subscribe mechanisms through web push techniques, so that any computing device that runs a web server can be a subscriber that is notified through HTTP POST requests. The Å publish/subscribe framework [37] was proposed following a content-based subscription scheme where event patterns can be defined using scripts with many common script languages supported. The RESTful paradigm is used together with a client event cache so that clients can periodically query the HTTP endpoint. PubSubHubBub [38] is a protocol for publish/subscribe messaging on the Internet, which extends Atom [39] and really simple syndication (RSS) [40] protocols for data feeds. It provides push Atom/RSS update notifications instead of requiring clients to poll the whole feeds. Constrained application protocol (CoAP) [41] is a RESTful application layer protocol that is intended for use in resource-constrained nodes and networks. Through the Observe option [42], a client can conditionally observe a resource on a CoAP server, only being informed about state changes meeting a specific condition or set of conditions.

2.2. Event-Driven Service-Oriented Architecture

The SOA paradigm has been adopted in different application domains, as it supports modularization of service components to be reused in heterogeneous platforms, especially using web services technology [43]. Traditional web services use a point-to-point mechanism between service providers and service consumers, where service consumers normally request some information from service providers, and service providers provide the requested information in a synchronous manner. From this perspective, service providers become information providers. When the number of services increases drastically, this approach can potentially create management and integration issues, especially when the information providers are everyday objects.

Event-driven architecture (EDA) [44] defines a methodology for designing and implementing applications and systems in which events are transmitted between decoupled software components and services. Information captured from objects can be treated as events, which may trigger some operations in different services. If a point-to-point approach is used, an event should be propagated to all services that are interested in using it, and the event source is in charge of carrying out this task. A brokered approach is better when the number of services that are interested in a specific event grows significantly; a publish/subscribe mechanism fits well to fill the void.

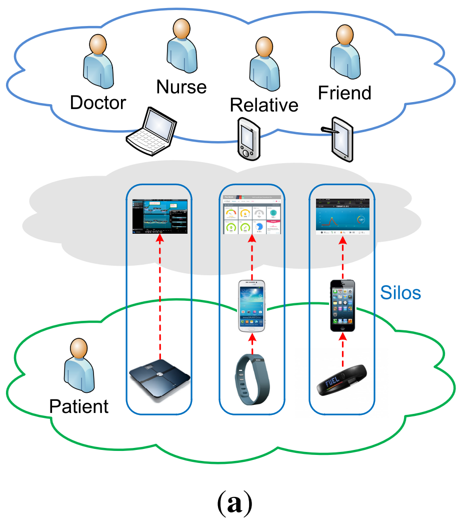
The enterprise service bus (ESB) [45] concept combines EDA and SOA approaches to simplify integration tasks, bridging heterogeneous platforms and environments. It facilitates interactions between service providers and service consumers, both in synchronous and asynchronous manners. There is no standardized specification for ESB implementations. In general, it should support message routing, message transformation, protocol mediation and event handling. Many ESB implementations provide various sophisticated functionalities for integrating new and legacy services and information systems. However, many of these functionalities require demanding programming efforts, which make ESB as an integration platform require quite a steep learning curve.

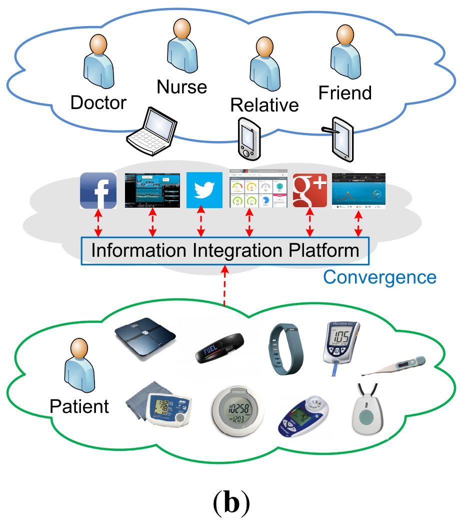
In this paper, a publish/subscribe platform and its working prototype is presented, providing information through information channels for event-driven information dissemination from everyday objects to Internet-based services. The platform itself is designed to be flexible enough, following the SOA paradigm, to accept not only information from physical devices, but also from any type of information provider. It aims to simplify the information integration process, while still following the event-driven SOA concept as ESB does, so that the platform can be used out-of-the-box without modifying anything in the platform programmatically. This is mainly achieved by imposing strong constraint that both service providers and service consumers should exchange messages through RESTful web services. Object gateways are responsible for encapsulating captured information from the objects as HTTP requests to be sent to the platform in case the objects cannot send HTTP requests directly. An overview of the platform's functionalities are discussed in the next section.

3. Conceptual Design

The IIP aims to bridge the communications between everyday objects (*i.e.*, information providers) and Internet-based services (*i.e.*, information consumers), acting as a service broker between the two entities. This broker is expected to break the information reusability issue in a vertical "silo" integration approach that has been chosen by many personal wearable device vendors, including within the healthcare sector; see Figure 1a.

Figure 1. (**a**) Point-to-point "silo" integration; (**b**) Brokered converged integration.





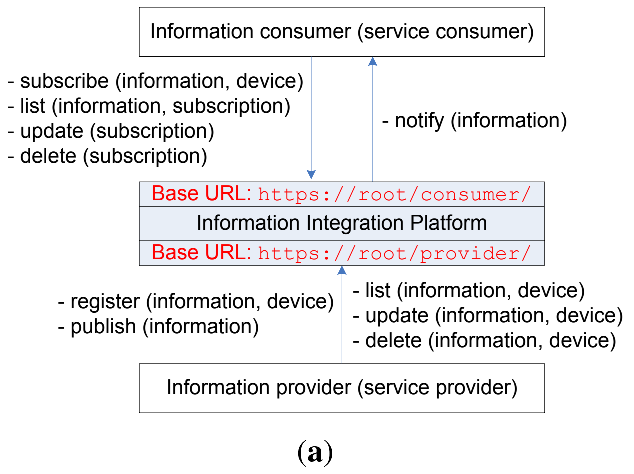
Patients are faced with various healthcare-related devices from different vendors in their daily activities, and the information these devices gather is commonly only used by specific services provided by the devices' vendors. Other Internet-based services have no or very limited possibilities to utilize such information, so it is common that similar information is redundantly gathered by different devices for their own services. This tight coupling between devices and services can be solved if device vendors provide open APIs that enable service developers to make use of the gathered information in their services. However, the integration normally still follows a point-to-point approach, as shown in Figure 1a, where each service is directly communicating with each device that provides the information. The downside of such point-to-point integration is, from the information providers' (*i.e.*, device) perspective, that they have to send newly gathered information to different services that are interested in using it. This is particularly an issue for battery-powered wireless devices. A brokered approach, as shown in Figure 1b, tackles this issue by delegating the information distribution task to the broker, so that information providers only need to send newly collected information once to the service broker. The service broker provides convergence for information gathering from various different devices that the patients encounter.

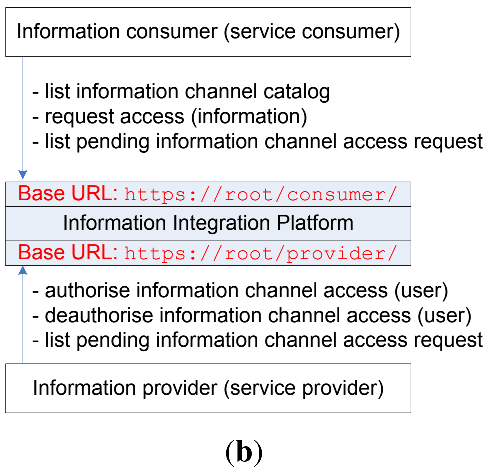
3.1. General Architecture

The IIP, which plays the service broker role, primarily aims to be deployed in the cloud for global reachability, although it is possible to deploy it in a closed environment, such as in smart homes.

RESTful web service interfaces are used facing both information providers (*i.e.*, devices) and information consumers (*i.e.*, Internet-based services), as they are widely used within the web domain. Figure 2a shows the main functionalities of the IIP.

Figure 2. (**a**) Main functionalities of the information integration platform (IIP); (**b**) Additional functionalities of the IIP.

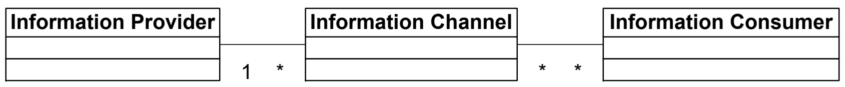

Information consumer (service consumer)
- subscribe (information, device)
- list (information, subscription)
- update (subscription)
- delete (subscription)
- notify (information)
Base URL: https://root/consumer/
Information Integration Platform
Base URL: https://root/provider/
- register (information, device)
- publish (information)
- list (information, device)
- update (information, device)
- delete (information, device)
Information provider (service provider)
(a)


Information consumer (service consumer)
- list information channel catalog
- request access (information)
- list pending information channel access request
Base URL: https://root/consumer/
Information Integration Platform
Base URL: https://root/provider/
- authorise information channel access (user)
- deauthorise information channel access (user)
- list pending information channel access request
Information provider (service provider)
(b)

In general, the IIP provides RESTful web service interfaces for both information providers (service providers from the SOA standpoint) and information consumers (service consumers from the SOA perspective) that manage the information flow from information providers to information consumers. These resources are divided into two categories with two main web resource-based uniform resource identifiers (URIs), namely https://root/provider/ and https://root/consumer/. The root part of the URI refers to the domain of the specific IIP deployment. Information is managed in different information channels, and information providers should initially create information channels before being able to pass through information they gather. The IIP provides resources for information providers to register new information channels with varying parameters, to list their information channels, to update/modify their information channels, to publish new information to their information channels and to delete/remove their information channels. In addition, the IIP provides resources for information consumers to list existing information channels, to subscribe to existing information channels, to update/modify their subscriptions to existing information channels and to delete/remove their subscriptions to existing information channels. When an information consumer subscribes to an existing information channel, it should provide a notification uniform resource locator (URL), which acts as an end-point for IIP to deliver notifications of newly published information from an information provider who owns that particular information channel. An information provider can register many information channels, but an information channel can only be associated with one information provider (*i.e.*, the owner). This makes the relationship between information provider and information channel a one-to-many relationship. On the other hand, an information channel can be accessed/subscribed to by many information consumers, and an information consumer can access/subscribe to many information channels. This makes the relationship between information channel and information consumer a many-to-many relationship. These relationships are shown in Figure 3. An article, which was written earlier by the author, describes how these functionalities work in a more detailed manner [46], including sequence diagrams and the contents of message exchanges. This work is a continuation of the author's previous work, addressing several aspects in the future work section of the previous article.

In addition to the main functionalities as previously described, the IIP provides resources for access control between different credentials within the IIP, as shown in Figure 2b.

Figure 3. Relationships between the information provider, information channel (in the IIP) and information consumer.



Since the IIP acts as a broker between information providers and information consumers, dependability becomes a crucial aspect that should be investigated. There are several definitions of dependability, but in general, it is commonly recognized as an integrative concept that encompasses different attributes. Littlewood and Strigini [47] suggested that dependability attributes include reliability, safety, security and availability. Avižienis *et al.* [48] defined attributes of dependability to comprise availability, reliability, safety, confidentiality, integrity and maintainability. In this latter view, security is not seen as a standalone attribute, but rather as a combination of three attributes, namely confidentiality, integrity and availability. This is in line with the confidentiality, integrity and availability (CIA) triad model [49], which commonly acts as a fundamental guideline to help secure information systems by providing a measurement tool for security implementations. The following subsections will cover the dependability aspects of the IIP that relate to security and availability, including scalability.

3.2. Security and Privacy

Information is passed around between information providers and information consumers through the IIP, and thus, communications between the three entities should be secured. Since REST interfaces are used in IIP, the communications security relies heavily on the application layer protocol used by these REST interfaces. In contrast to traditional web services technology that has a solid standardized security stack, such as WS-Security [50], RESTful web services do not have predefined and/or standardized security methods. Although RESTful web services were initially designed to be technology-agnostic [18], it has been commonly associated with the HTTP protocol. Thus, many of its security features are mainly inherited or simply adopted from the ones used for HTTP-based applications. Transport layer security (TLS), in the form of HTTP secure (HTTPS), has been the main ground for RESTful web services security, which provides a secure point-to-point communications channel on top of the transport layer.

The HTTP basic access authentication scheme, which was initially specified in the HTTP/1.0 specification, provides a simple authentication mechanism to access resources on a web server (based on URIs) by means of username and password. This scheme is not considered to be a secure method of user authentication, as the username and password are transmitted through the network as plain text (unless used in conjunction with other secure transport mechanism, such as HTTPS). Nevertheless, the combination of HTTPS and HTTP basic authentication in many cases is enough for securing resources on a web server, as everything being sent through the wire is encrypted.

From IIP's perspective (as shown in Figure 2a), information providers are applications that relay information from devices used by patients to information channels in the IIP. Likewise, information consumers are applications that consume/use information by subscribing to information channels in the IIP. To strengthen the privacy of information being exchanged between different applications through the IIP, access control to information channels should be maintained by the IIP. Identity-based access control is utilized, and an access control matrix is maintained. Information channels are registered by information providers (*i.e.*, the owners), and each information channel has exactly one owner who can publish information (add new information) to the registered information channel (write access to the information channel). The access control matrix is used for authorizing information consumers to access/subscribe to information channels (read access to information channels). An information provider has the privilege of specifying which credentials (e.g., usernames in HTTP basic authentication scheme) have read access (*i.e.*, can subscribe) to information channels it owns. Table 1 shows an example of an access control matrix maintained by the IIP.

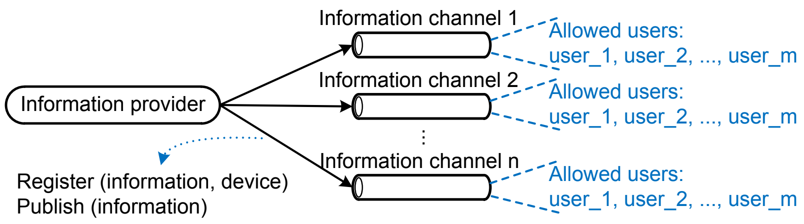
Table 1. Access control matrix for read access to information channels.

Usernames	Channel_1	Channel_2	...	Channel_n
username_1	YES	YES		NO
username_2	NO	YES		YES
⋮				
username_m	NO	NO		YES

The rows in Table 1 represent the capabilities of users (information consumers) in the IIP, and the columns represent access control lists of information channels. Information consumers can list all information channels (by utilizing a catalog service provided by the IIP), but they can only access/subscribe to information channels listed in their capabilities lists. Since one of the IIP's main responsibilities is to manage information channels, the access control matrix can be simplified to access control lists only (*i.e.*, the columns in Table 1). Whenever an information provider adds/removes a user (*i.e.*, an information consumer) from/to its allowed user list of a specific information channel it owns, the IIP will update the access control list of the corresponding information channel. Figure 4 shows the relationship between an information provider and its registered information channels with their allowed user lists. The IIP provides additional functionalities, as shown in Figure 2b, for managing access to information channels. Information consumers are provided with resources for listing information channels (information channel catalog service), requesting access to information channels and listing their pending access requests to information channels. Information providers, on the other hand, are provided with resources for authorizing access to information channels that belong to them (adding allowed users to their allowed user lists), deauthorizing access to information channels that belong to them (removing allowed users from their allowed user lists) and listing pending access requests to their information channels.

The identity-based access control to different information channels is meant to be used by applications (both information providers and information consumers). User identities in this case are application identities, not users of the applications (e.g., patients, nurses). These identities are used for controlling what information different applications can or cannot use, where the information is generated by other applications. However, these identities, which are used in the IIP, can also be directly mapped to user identities in the information consumer applications if required (e.g., a patient ID as a username in the IIP). Finer degrees of access control can be implemented by information consumer applications, for example, utilizing a role-based access control (RBAC) to group individual users of the applications.

Figure 4. Allowed user lists of information channels belonging to an information provider.



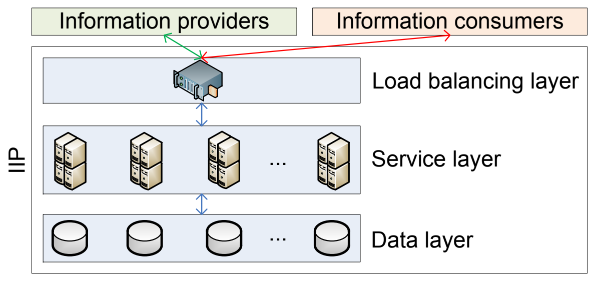
3.3. High Availability and Scalability

Despite the positive aspects of the IIP as an integration platform between things and services that provides information distribution convergence, as shown in Figure 1b, centralized service brokers, such as the IIP, are architecture-wise a single point of failure, since all information providers and information consumers communicate with and rely on it. High availability becomes a crucial factor for successful deployment of the IIP to ensure services receive information from devices in a timely manner and continue to work properly (*i.e.*, reliable). High availability is not a new topic in itself, as typical client-server systems require servers to run 24/7, with uptime as close to 100% as possible, accepting requests from client applications. Redundancy is the key to high availability, where service components are duplicated in different nodes, so that if one service component fails, another similar component will take over its tasks. In general, high availability can be achieved in either master/slave or master/master mode. In master/slave mode, a server instance (*i.e.*, the master) is in charge of providing services to the clients' requests, while another server instance (*i.e.*, the slave) is running idle. When the master instance fails, a monitoring entity (*i.e.*, the manager) will hand over the master's tasks to the slave instance. On the contrary, all server instances are treated as masters in master/master mode, all providing services to client requests. The manager is responsible for monitoring and handling any conflict that might arise between concurrent changes made by different master instances. A load balancer can be added in front of master instances, so that client requests can be distributed according to the processing capability of the master nodes (e.g., requests are evenly distributed among master nodes when they have similar processing capability).

The IIP utilizes master/master mode for high availability with an additional load balancer as proxy for handling client requests (both from information providers and consumers). This will make the IIP seem to be a single entity from both the information providers' and consumers' perspectives, but its components are redundantly distributed among different nodes. This approach allows the scalability aspect to be incorporated, as well, enabling new nodes to be added when the current serving nodes are reaching their peak (*i.e.*, fully loaded) in handling incoming requests. Scalability becomes important when the number of information providers and consumers using the IIP is not fixed. IIP nodes should be flexible enough to scale horizontally by the addition of new commodity servers when the number of participating information providers and consumers grows.

With regard to high availability and scalability, a simple three-layer system architecture is used by the IIP for deployment, as shown in Figure 5.

Figure 5. Three-layer system architecture for IIP deployment.



The load balancing layer acts as a proxy service, where both information providers and consumers send requests. The requests are then forwarded to one of the application servers in the service layer that hosts the main logic of the IIP (*i.e.*, the IIP application) based on the load balancing criteria maintained by the load balancer. In the service layer, the IIP's functionalities, as shown in Figure 2a,b, are realized and exposed through RESTful web service interfaces. All information that needs to be stored, such as information channels, allowed user lists, information channel subscriptions and the actual information of the information channels, are persisted in the data layer. By adopting this three-layer architecture, the IIP's components can be made redundant and can be scaled up according to deployment needs (e.g., add application nodes when more processing capability is needed, add data nodes when bigger storage capacity is required).

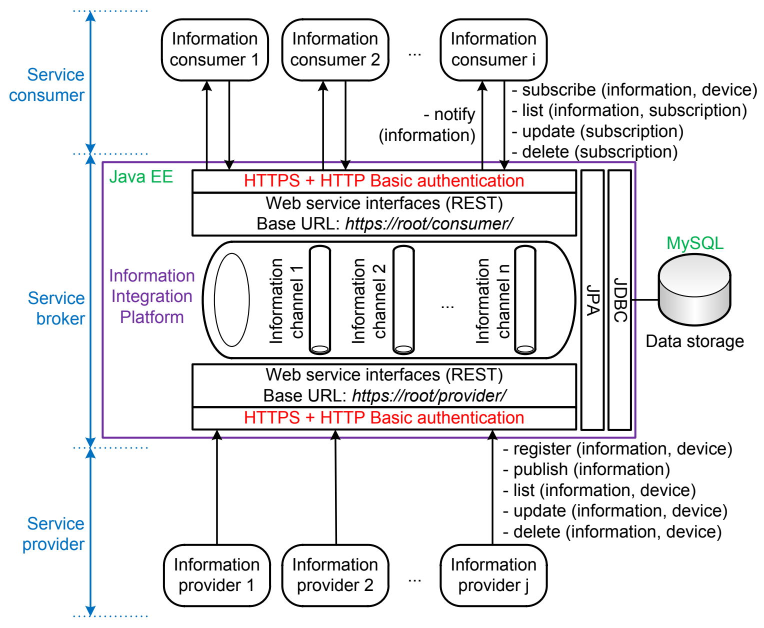
4. Prototype Implementation

The IIP and several service prototypes within the healthcare domain have been implemented following the conceptual design described in the previous section by utilizing open source software. This section will describe the prototype implementations that have been conducted as proof-of-concept of the conceptual design.

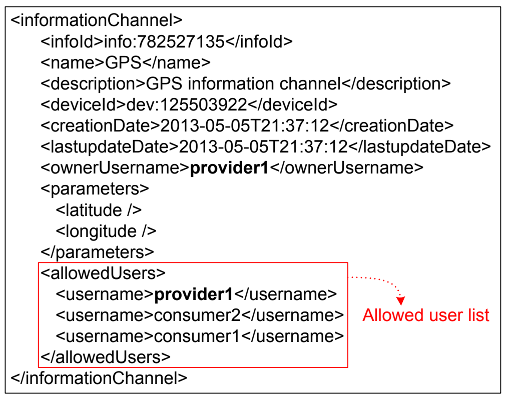
4.1. The IIP Prototype

The current prototype of the IIP has been implemented as a Java Enterprise Edition (EE) 6 application in the service layer, following Figure 5, deployed on the Glassfish 3.1.2.2 open source application server. The open source MySQL Cluster 7.3.2 is used for data storage in a clustered environment for high availability and scalability in the data layer, and the Java Persistence API (JPA) is utilized for mapping relational tables in the database to entity objects in the application through Java Database Connectivity (JDBC). EclipseLink 2.3.2 (JPA 2.0) is used as the JPA provider. HTTPS is employed for encrypting all message exchanges between the IIP and both information providers and consumers, and HTTP basic authentication is utilized for simple authentication to access the exposed web resources in the IIP. Figure 6a shows the prototype implementation architecture of the IIP.

Figure 6. (**a**) The IIP prototype implementation architecture; (**b**) Information channel representation example in extensible markup language (XML) format.


Service consumer
Information consumer 1
Information consumer 2
...
Information consumer i
- notify (information)
- subscribe (information, device)
- list (information, subscription)
- update (subscription)
- delete (subscription)
Java EE
HTTPS + HTTP Basic authentication
Web service interfaces (REST)
Base URL: https://root/consumer/
Service broker
Information Integration Platform
Information channel 1
Information channel 2
...
Information channel n
JPA
JDBC
MySQL
Data storage
Web service interfaces (REST)
Base URL: https://root/provider/
HTTPS + HTTP Basic authentication
- register (information, device)
- publish (information)
- list (information, device)
- update (information, device)
- delete (information, device)
Service provider
Information provider 1
Information provider 2
...
Information provider j

(**a**)



```
<informationChannel>
    <infoId>info:782527135</infoId>
    <name>GPS</name>
    <description>GPS information channel</description>
    <deviceId>dev:125503922</deviceId>
    <creationDate>2013-05-05T21:37:12</creationDate>
    <lastupdateDate>2013-05-05T21:37:12</lastupdateDate>
    <ownerUsername>provider1</ownerUsername>
    <parameters>
        <latitude />
        <longitude />
    </parameters>
    <allowedUsers>
        <username>provider1</username>
        <username>consumer2</username>
        <username>consumer1</username>
    </allowedUsers>
</informationChannel>
```

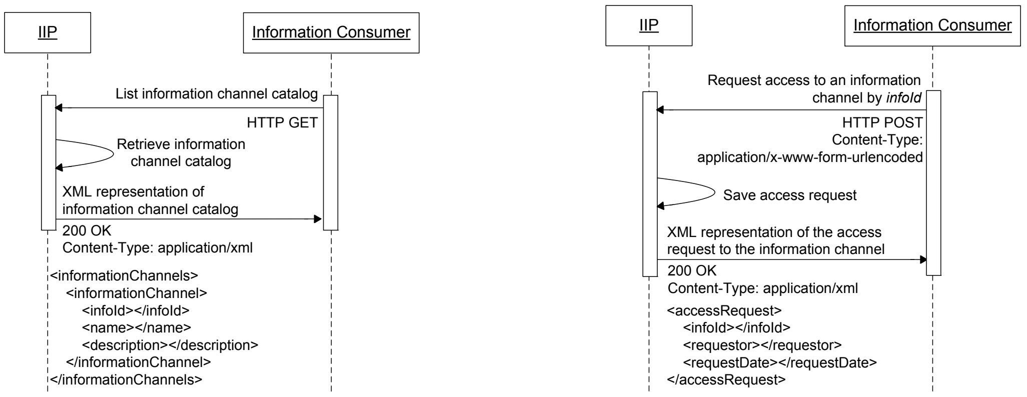
Allowed user list

(**b**)

4.1.1. Information Channel Access Management

Information channels are managed internally by the IIP, and they can be created (registered), modified (updated) and deleted (removed) dynamically through the provided REST interfaces without having to recompile and redeploy the application. In the current prototype, the representation of an information channel can be retrieved in extensible markup language (XML) format, as depicted in Figure 6b. Each information channel maintains its own allowed user list, and the HTTP basic authentication's credentials are directly used for accessing information channels. As described in the previous section, these credentials are used by client applications to authenticate themselves when communicating with the IIP. IIP users are categorized into three groups, namely *admin*, *provider* and *consumer*. Admin users are mainly responsible for adding, removing and assigning groups to users of the IIP. Users in the provider group are by default added to the consumer group, since information providers should be able to consume their own information (acting as information consumers). When an information provider creates (registers) a new information channel, its own username is added by default to the information channel's allowed user list as the first information consumer that is allowed to access the information in the information channel. Users in the consumer group that are not included in the provider group are strictly information consuming-only users, and they cannot create (register) new information channels. Information consumers can retrieve a list of available information channels through the information channel catalog service provided by the IIP at URL https://root/consumer/catalog/, which can be requested by using HTTP GET, as shown in Figure 7a. Only the *infoId*, *name* and *description* of information channels are returned in the catalog, while the ownerships are not revealed. Information consumers can request access to information channels (read-only) by sending HTTP POST to URL https://root/consumer/authorization/{infoId}/, where *infoId* is a unique ID of an information channel. Figure 7b shows a sequence diagram of an information consumer requesting access to an information channel.

Figure 7. (**a**) Information channel catalog request sequence diagram; (**b**) Information channel access request sequence diagram.


IIP
Information Consumer
List information channel catalog
HTTP GET
Retrieve information channel catalog
XML representation of information channel catalog
200 OK
Content-Type: application/xml
IIP
Information Consumer
Request access to an information channel by *infoId*
HTTP POST
Content-Type:
application/x-www-form-urlencoded
Save access request
XML representation of the access request to the information channel
200 OK
Content-Type: application/xml

```
<informationChannels>
  <informationChannel>
    <infoId></infoId>
    <name></name>
    <description></description>
  </informationChannel>
</informationChannels>
```

```
<accessRequest>
  <infoId></infoId>
  <requestor></requestor>
  <requestDate></requestDate>
</accessRequest>
```

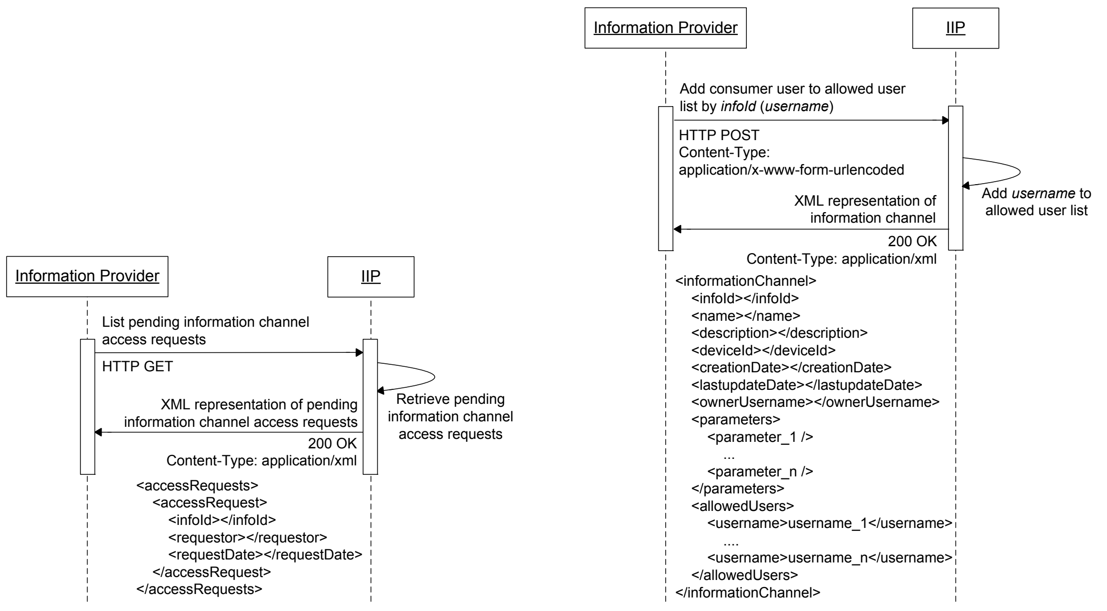
(**a**) (**b**)

Access requests to information channels are stored by the IIP, and a resource for listing pending access requests is made available to information providers through URL https://root/provider/authorization/, which can be accessed via an HTTP GET request. This resource will return all pending access requests to information channels belonging to the caller (*i.e.*, the information provider that accesses this resource) in XML format, as shown in Figure 8a. Information providers can add information consumer users to their information channels' allowed user lists (for read access) by sending HTTP POST request to a resource provided by the IIP at URL https://root/provider/authorization/{infoId}/. The content of this request should be of type application/x-www-form-urlencoded and contains a parameter called *username*, which refers to the information consumer user being added to the allowed user list. This resource will return a representation of the affected information channel in XML format, as shown in Figure 8b.

4.1.2. High Availability and Scalability

In order to provide a high availability of service, the IIP must run multiple redundant instances of service-providing entities, so that, if one instance stops functioning, other instances can still serve client requests. In relation to high availability, the IIP must also be able to scale to larger deployments in order to accommodate an increasing number of clients using its service. Application server clustering, which is supported by the Glassfish application server, can address the needs of both high availability and scalability. All application instances in a cluster, which can reside in different hosts, can be administered as a single unit, and user sessions can be automatically replicated between application instances in a cluster. However, following REST principles, the services being provided by the IIP are stateless, so no client session from each request is maintained by the IIP. All server resource states are persisted in the database. From this point of view, clustering at the service layer (following Figure 5) does not give much advantage, except simpler administration.

Figure 8. (**a**) Pending information channel access requests listing sequence diagram; (**b**) Adding an information consumer user to the allowed user list of an information channel sequence diagram.


Information Provider
IIP
List pending information channel access requests
HTTP GET
Retrieve pending information channel access requests
XML representation of pending information channel access requests
200 OK
Content-Type: application/xml
Information Provider
IIP
Add consumer user to allowed user list by *infoId* (*username*)
HTTP POST
Content-Type: application/x-www-form-urlencoded
Add *username* to allowed user list
XML representation of information channel
200 OK
Content-Type: application/xml

```
<accessRequests>
   <accessRequest>
      <infoId></infoId>
      <requestor></requestor>
      <requestDate></requestDate>
   </accessRequest>
</accessRequests>
```

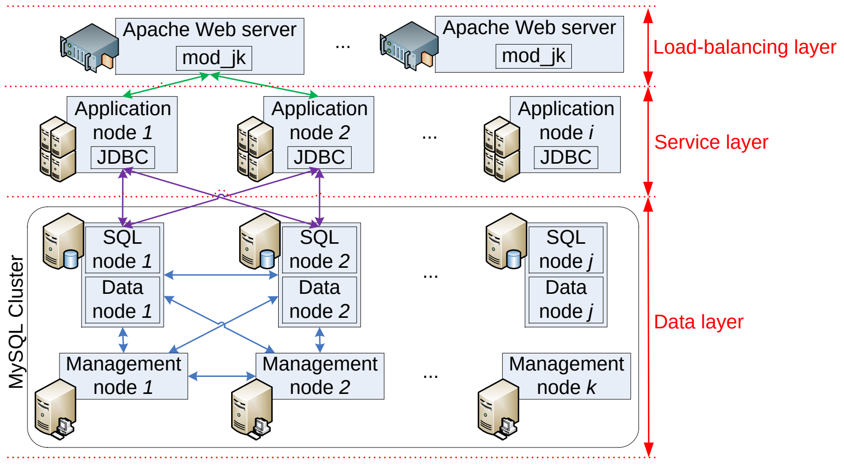
```
<informationChannel>
   <infoId></infoId>
   <name></name>
   <description></description>
   <deviceId></deviceId>
   <creationDate></creationDate>
   <lastupdateDate></lastupdateDate>
   <ownerUsername></ownerUsername>
   <parameters>
      <parameter_1 />
      ...
      <parameter_n />
   </parameters>
   <allowedUsers>
      <username>username_1</username>
      ....
      <username>username_n</username>
   </allowedUsers>
</informationChannel>
```

(**a**) (**b**)

The open-source Apache web server with the mod_jk module is used for load balancing requests coming from clients (both information providers and consumers, as shown in Figure 5), acting as a proxy. The load balancing factor is currently set as equal for all IIP instances (hosted in different hosts), so that requests are forwarded equally (*i.e.*, evenly distributed) among all IIP application instances. New IIP application instances in different hosts can be added (scaled up) in the service layer and the load balanced through this proxy.

In the data layer (following Figure 5), MySQL Cluster is used for storing all of IIP's information. The IIP application instance in the service layer is responsible for handling requests from clients, but it does not store any state or information. Instead, it communicates with the back-end database to store information. MySQL Cluster follows a master/master architecture with no single point of failure, providing high availability and scalability of data storage and access. It makes use of a specialized storage engine, called *NDBCLUSTER*, which is not used in the standard MySQL server, and employs a synchronous replication to guarantee that data is written to multiple nodes upon committing the data. Unlike in the service layer, data replication in the data layer is essential to maintain high availability of data, which, in turn, will make the IIP run properly (*i.e.*, reliable). Three node types of MySQL Cluster are used in the prototype, namely *management*, *data* and *SQL*. Management nodes are utilized for managing the entire cluster. Data nodes are mainly responsible for storing and retrieving data from memory and disk. SQL nodes are used for providing application access to the cluster. Application instances communicate with the MySQL Cluster through the SQL nodes by using JDBC, which is responsible for load balancing queries across the SQL nodes. Figure 9 shows the current prototype's deployment architecture to accommodate high availability and scalability aspects.

Figure 9. High-available and scalable IIP prototype deployment.

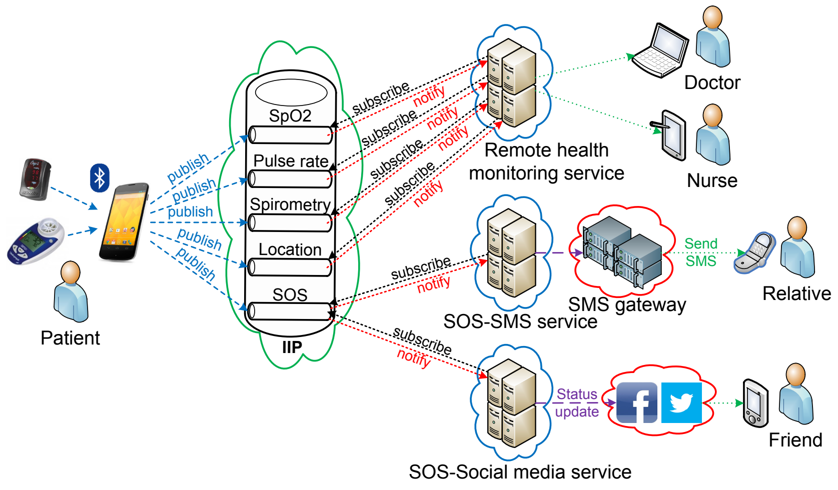


The current prototype consists of seven hosts (virtual machines) for deploying different nodes of the IIP: one for the load balancer, two for application nodes, two for data and SQL nodes and two for management nodes. By following the deployment architecture in Figure 9, new nodes can be added when needed.

4.2. Healthcare Services Prototype

Three prototype services have been implemented within the healthcare domain for patient-centric well-being as proof-of-concept. The services are developed as information consumers that make use of information from the IIP. Figure 10 shows an end-to-end perspective of the developed service prototypes. Information is gathered from several devices relayed through an application gateway running on an Android smartphone and forwarded to the IIP. All implemented services subscribe to information channels of interest and provide services to different healthcare actors.

Figure 10. End-to-end perspective of implemented service prototypes.

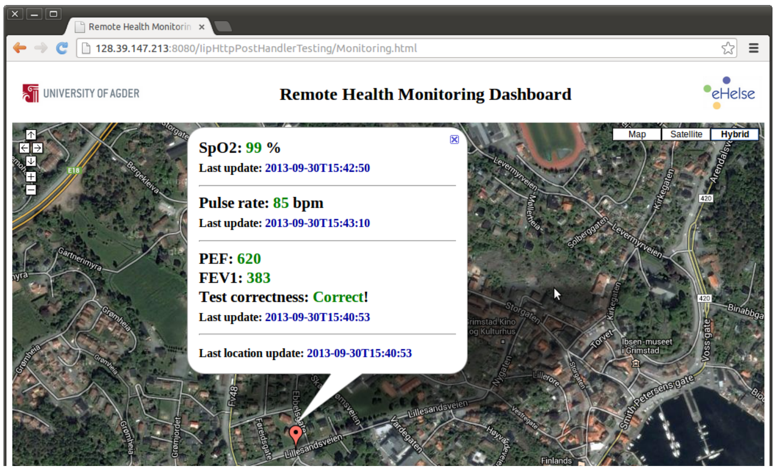


Two commercial off-the-shelf wireless Bluetooth medical devices are currently used for gathering the vital signs (*i.e.*, Nonin Onyx II 9560 for blood oxygen saturation (SpO2) and pulse rate data, Vitalograph copd-6 bt for spirometry data).

4.2.1. Remote Health Monitoring Service

The remote health monitoring service is implemented as a web application that is accessible by the users (e.g., doctors, nurses) via web browsers. It utilizes Java servlet technology for handling incoming HTTP POST notifications from the IIP and simple hypertext markup language (HTML) with asynchronous JavaScript and XML (AJAX) for presenting the output to the users. This service shows the latest measurements information from the remote patient, which currently includes pulse rate, blood oxygen saturation (SpO2), spirometry and location. The service subscribes to these four information channels and receives almost real-time notifications from the IIP. At the patient's side, an android application has been implemented for gathering measurements from a wireless pulse oximeter and spirometer devices through Bluetooth connections. This application sends all measurements to their corresponding information channels in the IIP. Location information is gathered directly by the Android application from the smartphone's built-in global positioning system (GPS) sensor. The web application's user interface is shown in Figure 11. By utilizing this service, healthcare personnel can monitor the patient's health condition remotely. Since mobile devices are used at the patient's side, the patient is not restricted to measurements from a specific location.

Figure 11. Remote health monitoring service user interface.



Remote Health Monitoring Dashboard

SpO2: 99 %
Last update: 2013-09-30T15:42:50

Pulse rate: 85 bpm
Last update: 2013-09-30T15:43:10

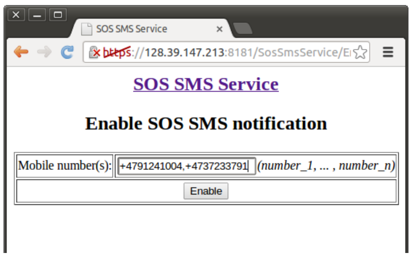
PEF: 620
FEV1: 383
Test correctness: Correct!
Last update: 2013-09-30T15:40:53

Last location update: 2013-09-30T15:40:53

4.2.2. SOS-SMS Service

This service provides emergency situation notifications via short message service (SMS) to specific recipients, enabling the patient to inform selected persons (e.g., relatives, friends, nurses) that he/she needs immediate help. The ideal interface to the patient would be a physical alarm button that can be easily pushed in case of emergency. For simplification, an Android application that mimics an alarm button is used in the current prototype. An SOS information channel is registered in the IIP, and a server-side application, based on Java servlet technology, is implemented to subscribe and to handle HTTP POST notifications from the IIP whenever the patient sends SOS messages. A simple web page is provided to the patient to administer which mobile numbers should be notified in case of emergency and also to enable or disable this service. Figure 12a shows a web page for the patient to enable emergency notifications via SMS to a list of mobile numbers, and Figure 12b shows a web page confirming that the service is enabled. When the patient enables this service, the web application subscribes to SOS information channel in the IIP for notifications of emergency events. In reverse, the web application unsubscribes from SOS information channel when this service is disabled by the patient.

Figure 12. (**a**) Web interface for enabling specific mobile numbers to be notified in the case of emergency; (**b**) Web interface confirming that SOS-short message service (SMS) is enabled.

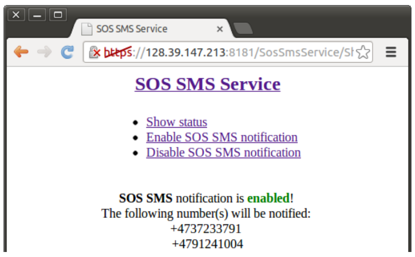


SOS SMS Service

Enable SOS SMS notification

Mobile number(s):	+4791241004,+4737233791	(number_1, ... , number_n)
Enable		

(**a**)



SOS SMS Service

- Show status
- Enable SOS SMS notification
- Disable SOS SMS notification

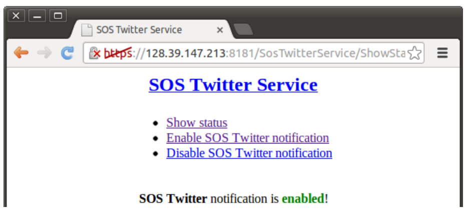
SOS SMS notification is enabled!
The following number(s) will be notified:
+4737233791
+4791241004

(**b**)

4.2.3. SOS-Social Media Service

This service is similar to the SOS-SMS service, except that it uses social media as the emergency notification medium. Social media have been used extensively in the last couple of years, and can be extended further to support emergency situations for the patient. People within the patient's social media circle can be the first responders in case of emergency (e.g., due to close proximity to the patient). A prototype that uses a Twitter account to disseminate emergency information has been implemented, utilizing the same SOS information channel used by the SOS-SMS service. The author's Twitter account is used in the current prototype. This service can be enabled or disabled by the patient through a web page similar to the SOS-SMS service (the two services are deployed as different applications). Figure 13a shows a web page confirming that the SOS-Twitter service is enabled, and Figure 13b shows a tweet of an emergency notification from the patient (*i.e.*, the author).

Figure 13. (**a**) Web interface confirming that SOS-Twitter service is enabled; (**b**) A tweet of an emergency notification.

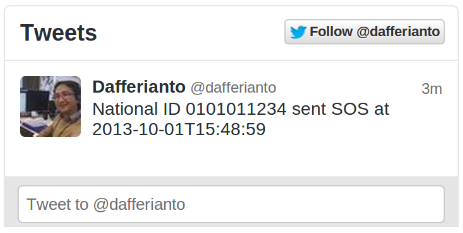


SOS Twitter Service

- Show status
- Enable SOS Twitter notification
- Disable SOS Twitter notification

SOS Twitter notification is enabled!

(**a**)



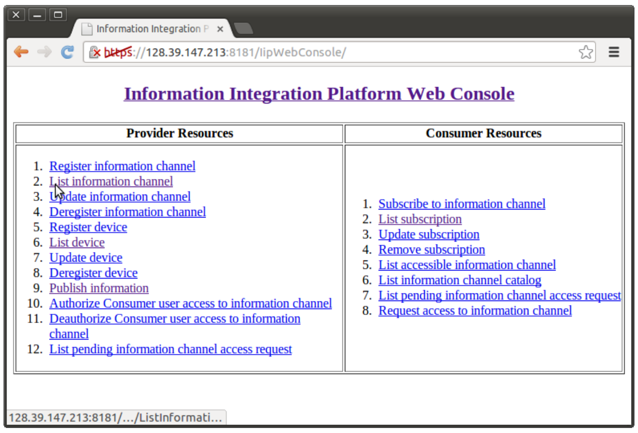
Dafferianto @dafferianto 3m
National ID 0101011234 sent SOS at 2013-10-01T15:48:59

(**b**)

4.3. The IIP Web Console Prototype

The IIP's functionalities are exposed as RESTful web services. While this can be seen as a positive way to enable brokered machine-to-machine (M2M) communications, human intervention is rather difficult to accommodate. Although communications between devices, the IIP and services should ideally be automated, human involvement is sometimes needed, especially during the development process of new services. A web console for the IIP has been implemented to ease the management of resources in the IIP (e.g., information channels, devices, subscriptions, access control) for both information providers and consumers with simple HTML pages. This web application utilizes the same REST interfaces that information providers and consumers use. Figure 14a shows the main page of the web console, which contains functionalities for both information providers and consumers. Figure 14b shows a web page for information providers to list their information channels. Each resource requires the user to enter HTTP basic authentication credentials like normal applications do.

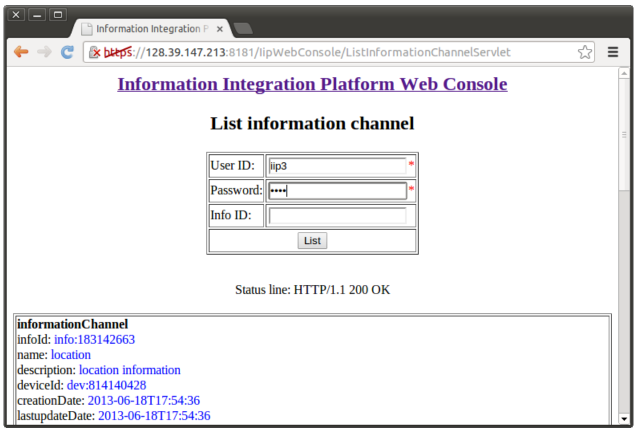
Figure 14. (**a**) The IIP web console's main page; (**b**) A web page for information providers to list their information channels.


Information Integration P
~~https~~://128.39.147.213:8181/IipWebConsole/

Information Integration Platform Web Console

Provider Resources	Consumer Resources
1. Register information channel 2. List information channel 3. Update information channel 4. Deregister information channel 5. Register device 6. List device 7. Update device 8. Deregister device 9. Publish information 10. Authorize Consumer user access to information channel 11. Deauthorize Consumer user access to information channel 12. List pending information channel access request	1. Subscribe to information channel 2. List subscription 3. Update subscription 4. Remove subscription 5. List accessible information channel 6. List information channel catalog 7. List pending information channel access request 8. Request access to information channel

(**a**)


Information Integration P
~~https~~://128.39.147.213:8181/IipWebConsole/ListInformationChannelServlet

Information Integration Platform Web Console

List information channel

User ID:	iip3 *
Password:	•••• *
Info ID:	

Status line: HTTP/1.1 200 OK

informationChannel
infoId: info:183142663
name: location
description: location information
deviceId: dev:814140428
creationDate: 2013-06-18T17:54:36
lastupdateDate: 2013-06-18T17:54:36

(**b**)

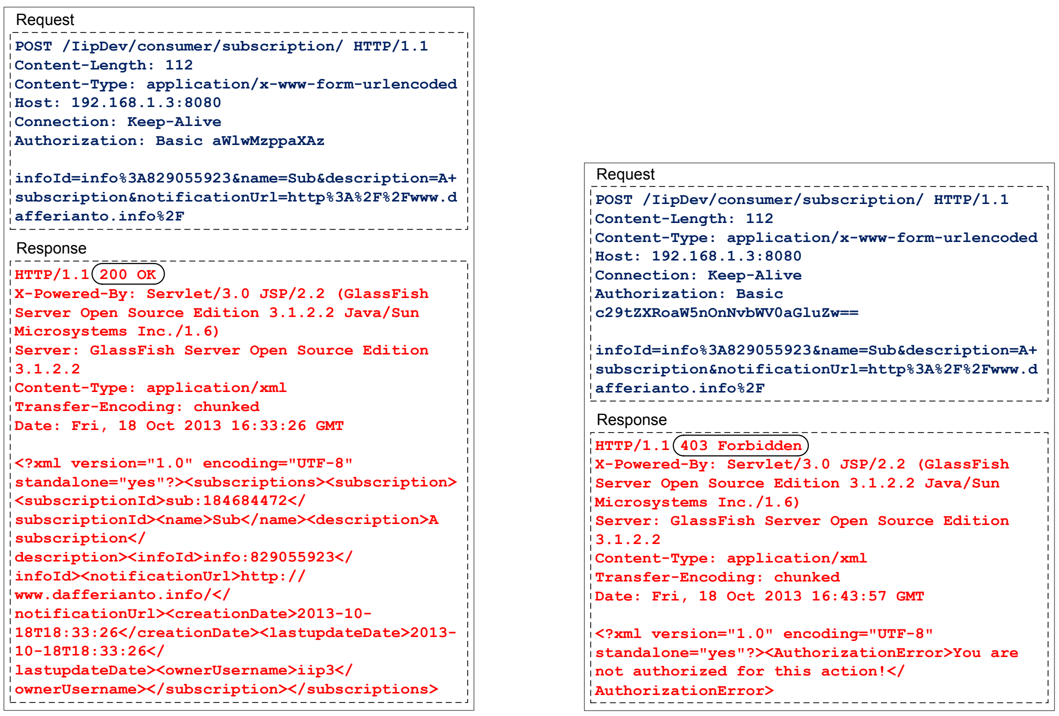
5. Evaluation

The implemented IIP prototypes, both standalone (*i.e.*, all components in one dedicated host) and clustered (*i.e.*, the components are spread in several different hosts), have been deployed and tested to work as intended in laboratory environment. This section will briefly present several aspects of the IIP that have been tested and verified, as well as performance benchmarking, which provide inputs on how to further improve the current implementation.

5.1. Information Channel Access Management

As described in the previous section, HTTPS is used for securing message exchanges between information providers, consumers and the IIP, by means of encryption. This will minimize the success rate of man-in-the-middle attacks. To maintain information privacy between applications, HTTP basic authentication credentials are directly used in IIP for authorizing applications to access different information channels. Information consumers that are not listed in an information channel's allowed user list will not be able to access or subscribe to that particular channel. Figure 15a shows a test case where an HTTP POST request is sent from a test client application, acting as an information consumer, to the IIP for subscribing to an information channel with *infoId* info:829055923. The username, which is used by the information consumer, is listed in the information channel's allowed user list, and thus, a 200 OK response is returned by the IIP, containing a representation of the subscription in XML format in its body. Figure 15b shows another test case where similar HTTP POST request for subscription to the same information channel is sent from an information consumer. This time, however, the username used by the information consumer is not listed in the information channel's allowed user list, and therefore, the IIP returns a 403 Forbidden response with an XML-formatted message in its body, informing that the username is unauthorized to subscribe to the targeted information channel. All messages in both tests (Figure 15a,b) are captured using Wireshark, and HTTPS is not used (otherwise all packets are encrypted and cannot be interpreted). It can be concluded from the conducted tests that the implemented information channel access management scheme by directly utilizing HTTP basic credentials works as intended.

Figure 15. Wireshark captures of: (**a**) An information consumer successfully subscribes to an information channel; (**b**) An information consumer is rejected when trying to subscribe to an information channel.



Request

```
POST /IipDev/consumer/subscription/ HTTP/1.1
Content-Length: 112
Content-Type: application/x-www-form-urlencoded
Host: 192.168.1.3:8080
Connection: Keep-Alive
Authorization: Basic aWlwMzppaXAz

infoId=info%3A829055923&name=Sub&description=A+
subscription&notificationUrl=http%3A%2F%2Fwww.d
afferianto.info%2F
```

Response

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/3.0 JSP/2.2 (GlassFish
Server Open Source Edition 3.1.2.2 Java/Sun
Microsystems Inc./1.6)
Server: GlassFish Server Open Source Edition
3.1.2.2
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Fri, 18 Oct 2013 16:33:26 GMT

<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><subscriptions><subscription>
<subscriptionId>sub:184684472</
subscriptionId><name>Sub</name><description>A
subscription</
description><infoId>info:829055923</
infoId><notificationUrl>http://
www.dafferianto.info/</
notificationUrl><creationDate>2013-10-
18T18:33:26</creationDate><lastupdateDate>2013-
10-18T18:33:26</
lastupdateDate><ownerUsername>iip3</
ownerUsername></subscription></subscriptions>
```

(**a**)

Request

```
POST /IipDev/consumer/subscription/ HTTP/1.1
Content-Length: 112
Content-Type: application/x-www-form-urlencoded
Host: 192.168.1.3:8080
Connection: Keep-Alive
Authorization: Basic
c29tZXRoaW5nOnNvbWV0aGluZw==

infoId=info%3A829055923&name=Sub&description=A+
subscription&notificationUrl=http%3A%2F%2Fwww.d
afferianto.info%2F
```

Response

```
HTTP/1.1 403 Forbidden
X-Powered-By: Servlet/3.0 JSP/2.2 (GlassFish
Server Open Source Edition 3.1.2.2 Java/Sun
Microsystems Inc./1.6)
Server: GlassFish Server Open Source Edition
3.1.2.2
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Fri, 18 Oct 2013 16:43:57 GMT

<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><AuthorizationError>You are
not authorized for this action!</
AuthorizationError>
```

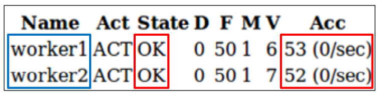
(**b**)

The tests were conducted manually, since they were only intended to verify whether the access management implementation worked correctly. Automated unit testing could be used instead for a more formal way of verification, and protocol conformance testing could be applied, as well, to find out whether the implemented features comply with the chosen standards.

5.2. High Availability and Load Balancing

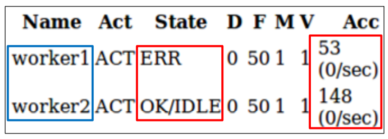
Seven hosts (machines) are used for deploying the clustered prototype of the IIP, where one is utilized as a proxy server for load balancing requests from both information providers and consumers, two are used for deploying the main applications that handle requests forwarded (load balanced) by the proxy/load balancer host, another two are employed for data and SQL nodes of the MySQL Cluster that act as the main storage for the IIP and the last two are utilized for management nodes of the MySQL Cluster. This set-up conforms with Figure 9 and can be considered as the minimum requirement for IIP's high availability, where all nodes have exactly one redundant backup (except the load balancer). New nodes can be added when deemed needed. A test was conducted to review the general functionality of the load balancing between the redundant nodes being deployed. A test client application was developed, playing the role of information provider that keeps sending one publication message per second to one of its information channels through the load balancer host. Initially, both application servers' hosts are up and running, and the load balancer host distributes the requests evenly to both application servers. As can be seen in Figure 16a, the load balancer host (acting as a proxy) successfully forwards all requests evenly to both application servers' hosts (*i.e.*, 53 and 52 requests for *worker1*and *worker2*, respectively). One of the two application servers' hosts was then turned off during the test, leaving only one application server available for handling all requests. From Figure 16b it can be seen that the load balancer host forwards all requests to the available application server's host (*worker2*), and *worker1*'s state was changed to error. From this test, it can be concluded that the redundancy of the application nodes works well for providing highly available service, while enabling new nodes to be added (horizontal scaling) and load balanced to serve incoming requests.

Figure 16. Snapshot of mod_jk status worker: (**a**) Both application nodes are up and running; (**b**) One application node (*worker1*) is down.



Name	Act	State	D	F	M	V	Acc
worker1	ACT	OK	0	50	1	6	53 (0/sec)
worker2	ACT	OK	0	50	1	7	52 (0/sec)

(**a**)

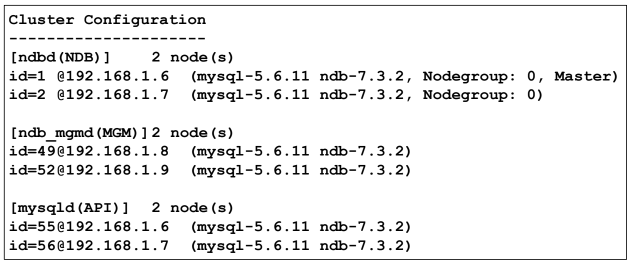


Name	Act	State	D	F	M	V	Acc
worker1	ACT	ERR	0	50	1	1	53 (0/sec)
worker2	ACT	OK/IDLE	0	50	1	1	148 (0/sec)

(**b**)

In the data layer, synchronous replication among data nodes is handled automatically by the MySQL Cluster, which is monitored and managed by the management nodes. Figure 17a depicts a snapshot of a MySQL Cluster management client that describes the most current configuration of the cluster and the status of each node. In this set-up, any data committed from the service layer is inserted into both data nodes synchronously. One of the two data and SQL nodes' hosts (with IP address 192.168.1.7) was shut down. The MySQL Cluster then saves all data and writes to only one existing data node, as shown in Figure 17b. When the host is up and running again, MySQL Cluster automatically synchronizes all changes to the newly running data node.

Figure 17. Snapshot of ndb_mgm client showing the most current MySQL Cluster configuration: (**a**) Both data and SQL node hosts are up and running; (**b**) One data and SQL node host (192.168.1.7) are down.

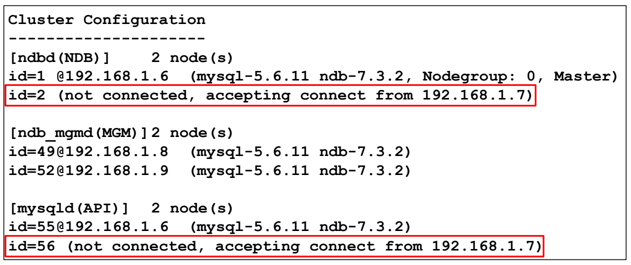


```
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1 @192.168.1.6  (mysql-5.6.11 ndb-7.3.2, Nodegroup: 0, Master)
id=2 @192.168.1.7  (mysql-5.6.11 ndb-7.3.2, Nodegroup: 0)

[ndb_mgmd(MGM)] 2 node(s)
id=49@192.168.1.8  (mysql-5.6.11 ndb-7.3.2)
id=52@192.168.1.9  (mysql-5.6.11 ndb-7.3.2)

[mysqld(API)]   2 node(s)
id=55@192.168.1.6  (mysql-5.6.11 ndb-7.3.2)
id=56@192.168.1.7  (mysql-5.6.11 ndb-7.3.2)
```

(**a**)



```
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1 @192.168.1.6  (mysql-5.6.11 ndb-7.3.2, Nodegroup: 0, Master)
id=2 (not connected, accepting connect from 192.168.1.7)

[ndb_mgmd(MGM)] 2 node(s)
id=49@192.168.1.8  (mysql-5.6.11 ndb-7.3.2)
id=52@192.168.1.9  (mysql-5.6.11 ndb-7.3.2)

[mysqld(API)]   2 node(s)
id=55@192.168.1.6  (mysql-5.6.11 ndb-7.3.2)
id=56 (not connected, accepting connect from 192.168.1.7)
```

(**b**)

5.3. Performance Benchmark

A dedicated multi-threaded client application was developed for performance benchmarking. The application acts as an information provider that constantly publishes information to the IIP with a predefined time interval for a certain period. Another client application, which subscribes to an information channel that belongs to the information provider application, was developed to receive notifications from the IIP, playing the role of information consumer. The main aim of this evaluation was to compare the performance of the standalone IIP prototype with the clustered version in terms of one complete flow of publication and notification average time. In addition, comparisons between the inclusion of the security and privacy scheme and also without it were conducted.

All experiments were carried out in a laboratory environment, where all hosts (machines) were deployed in an isolated network with a switch as the connecting point (there is no connection to an external network, such as the Internet). Experiments in a standalone set-up involve three hosts: one for the information provider application, one for the IIP (all-in-one, single point of failure) and one for the information consumer application. Experiments in a clustered set-up make use of nine hosts: one for the information provider application, seven for the IIP (as described earlier) and one for the information consumer application. All hosts that are used for deploying the IIP (both the standalone version and its clustered counterpart) have similar specifications (*i.e.*, Intel Core 2 Duo 2.4 GHz processor, 8 GB RAM running Linux Ubuntu 12.04 LTS). Another two hosts that deploy the information provider and consumer applications also have similar specifications (*i.e.*, Intel Core 2 Duo 2.4 GHz processor, 4 GB RAM running Linux Ubuntu 12.04 LTS). All application servers' configurations are kept similar with almost no optimization from their default settings to ensure fairness in the comparisons.

5.3.1. Publication Rate as a Variable

Four experiments were conducted in this category. All variables are fixed, except the publication rates of the information provider application, which were varied between one and 40 publications per second, and all publication and notification messages contain only one parameter. The first experiment applied the security measures (*i.e.*, HTTPS and HTTP basic authentication) in a standalone IIP set-up, while the second experiment did not incorporate any security mechanism, also in a standalone set-up. The third and fourth experiments were conducted in a clustered IIP set-up, where security was applied in the third experiment and was ignored in the fourth experiment. Each measurement in all experiments lasted five minutes, generating 300 to 12,000 data messages (depending on the publication rate). Each measurement was performed three times to ensure data consistency, and the first 1% of the captured data is removed from every measurement to avoid the start-up effect of the application servers in serving incoming requests. All measurement data were averaged and plotted alongside confidence intervals at the 95% confidence level.

Figure 18. Publication and notification time comparisons with publication rate as variable: (**a**) Standalone secure *vs.* insecure; (**b**) Clustered secure *vs.* insecure; (**c**) Secure standalone *vs.* clustered; (**d**) Insecure standalone *vs.* clustered.

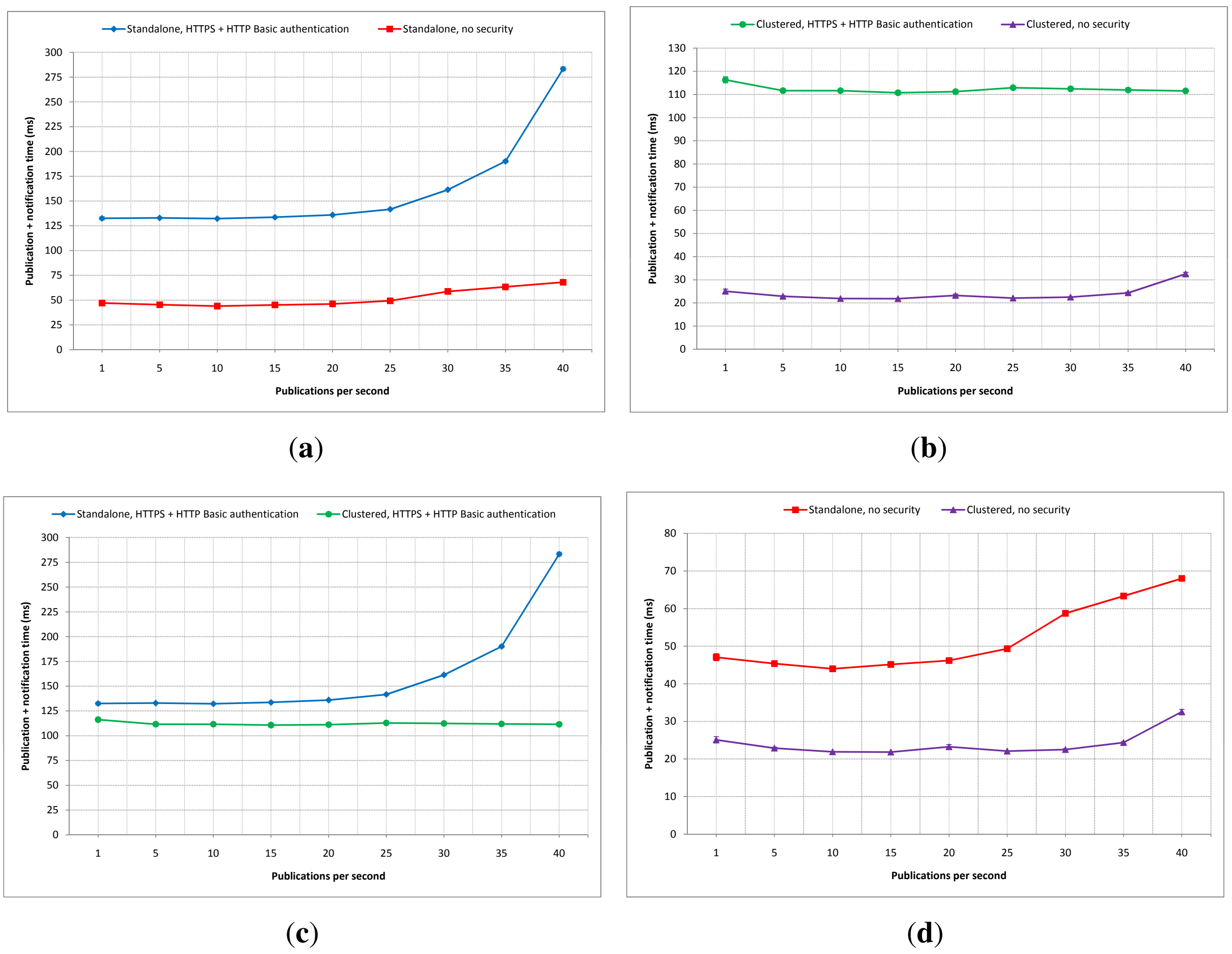


Figure 18a shows a comparison of total average publication and notification times between the secured and non-secured standalone IIP set-ups. It can be seen that the average difference at one publication per second is about 85 ms, which is the rough estimate of the security scheme's overhead in the standalone set-up. This average difference does not change much with the increase of the publication rate until around 30 publications per second. From there, the difference gap grows larger significantly.

Figure 18b shows a comparison of similar latency measurements as in Figure 18a, except that the compared experimental results are between secured and non-secured clustered versions of the IIP. In a clustered set-up, the starting difference between secured and non-secured implementations is about

90 ms, slightly higher than in the standalone set-up. This difference gap is maintained in a relatively stable manner up to 40 publications per second. From Figure 18a,b, it can be seen that the clustered version of the IIP can handle the increase of the publication rate better than its standalone counterpart, especially with the security scheme being applied.

A latency comparison between secured standalone and clustered IIP set-ups is depicted in Figure 18c. The average difference gap is stabled at about 20 ms up to around 25 publications per second, and then, it grows more than two times. This can be viewed as the lesser ability of the standalone IIP set-up in handling faster publication rates compared to the clustered set-up.

Figure 18d shows a latency comparison for publication and notification between non-secured standalone and clustered IIP set-ups. The average difference stays almost unchanged at about 20 ms from one to 25 publications per second, and the gap slightly widens as the publication rate increases.

From the four experiments conducted in this category (*i.e.*, publication rate as a variable), it can be concluded that the clustered deployment of the IIP handles higher rates of publications better than the standalone set-up. Even at lower rates, the total average latency for publication and notification are smaller in the clustered set-up, although only by a small margin. The inclusion of the security scheme adds additional overhead to the overall processing times in both standalone and clustered set-ups. The confidence intervals are very small compared to the mean values in all experiments. However, all experiments were carried out with only one parameter inside the publication and notification messages. In the next category of experiments, the number of parameters will be used as a variable.

5.3.2. Number of Parameters as a Variable

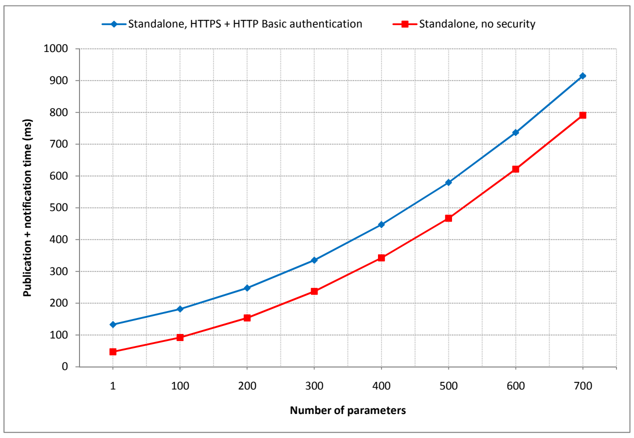
Experiments in this category were carried out to see how a different number of parameters affects the overall performance of the implemented platform in both standalone and clustered versions. In Figure 10, for example, the SpO2, the pulse rate and the SOS information channels use one parameter each; the location information channel has two parameters, while the spirometry information channel has 13 parameters for each measurement.

Four experiments were conducted in this category, but unlike the previous category, the publication rate is fixed at one publication per second in order not to load the application servers. The number of parameters is varied instead, ranging from one to 700 parameters. Each measurement in all experiments lasted five minutes, generating 300 data. Each measurement was performed three times to ensure data consistency, and the first 1% of the captured data is removed from every measurement to avoid the start-up effect of the application servers in serving incoming requests. All measurement data are averaged and plotted alongside their confidence intervals at the 95% confidence level, just like in the previous category.

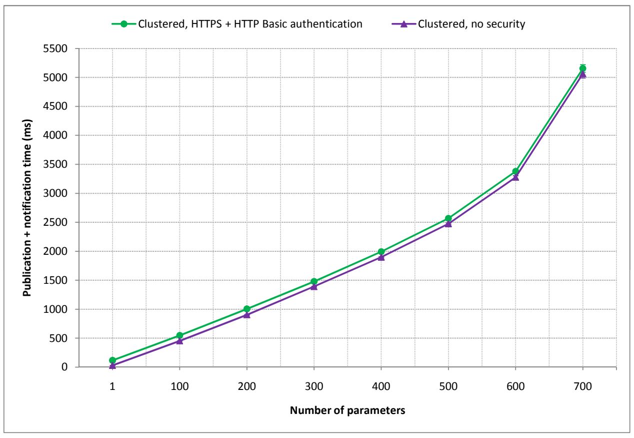
A comparison of total average publication and notification times between the secured and non-secured standalone IIP deployments is shown in Figure 19a. The time difference starts at about 85 ms when one parameter is used, and the gap increases almost linearly to around 125 ms when 700 parameters are used. This gap represents the implemented security mechanism's overhead in the standalone IIP set-up.

Figure 19b depicts a comparison of publication and notification latency between the secured and non-secured clustered IIP set-ups. The time difference is stabled across all measurements at about 90 ms. From Figure 19a,b, it can be seen that the additional overhead of the security mechanism does not change much with the increased number of parameters being used for both standalone and clustered deployments of the IIP.

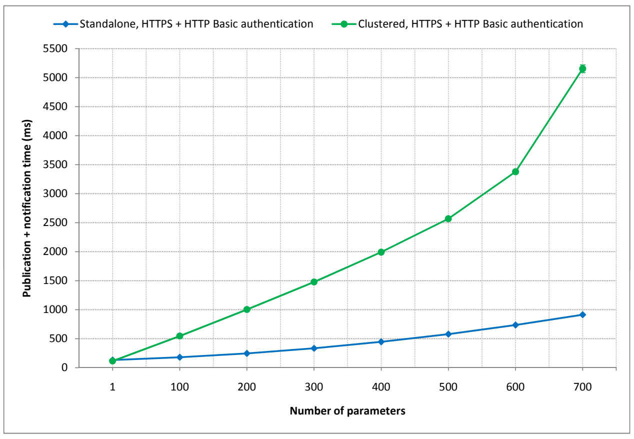
Figure 19. Publication and notification time comparisons with the number of parameters as the variable: (**a**) Standalone secure *vs*. insecure; (**b**) Clustered secure *vs*. insecure; (**c**) Secure standalone *vs*. Clustered; (**d**) Insecure standalone *vs*. clustered.



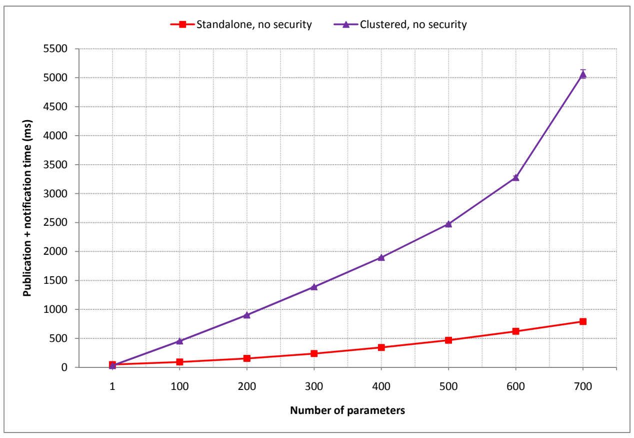
(**a**)



(**b**)



(**c**)



(**d**)

A comparison of the average latency difference between secured standalone and clustered IIP deployments is shown in Figure 19c. When only one parameter is used, the clustered set-up outperforms its a standalone rival with around a 20 ms difference. However, the clustered deployment suffers more latency overhead compared to the standalone version with the increasing number of parameters being used. With 100 parameters used, the clustered deployment performs worse than its standalone counterpart by around 365 ms, and it is further worsened as the number of parameters being used is increased.

Figure 19d shows a comparison of average publication and notification times between non-secured standalone and clustered IIP set-ups. Almost similar to Figure 19c, the clustered version in this experiment wins in terms of the latency difference compared to its standalone counterpart by about 20 ms when only one parameter is used, but it suffers when the number of parameters increases. When

100 parameters are used, the clustered set-up falls short by around 360 ms in latency performance compared to the standalone deployment.

From the four experiments in this category (*i.e.*, the number of parameters as the variable), it can be concluded that the clustered set-up of the IIP performs better than the standalone set-up when the number of parameters being used is low. The latency increase rate is higher for the clustered version compared to the standalone deployment as the number of parameters being used grows. This can be seen as a direct impact of the synchronization process between data nodes in the data layer, since synchronizing large amounts of incoming new data takes much more time than writing directly to one data store. The use of the security mechanism adds overhead to both standalone and clustered set-ups, but the processing times are not affected significantly with the increasing number of parameters being used. The confidence intervals are very small compared to the mean values in all experiments.

6. Conclusions and Future Work

An information integration platform (*i.e.*, the IIP) has been designed and developed to bridge communications between everyday objects and Internet-based services, breaking the traditional vertical "silo" approach of integration. This broker platform follows an event-driven SOA paradigm with a publish/subscribe messaging pattern and exposes its functionalities through a set of RESTful web services. An identity-based access control is used and has been implemented in the prototype to ensure information privacy between service clients (*i.e.*, information providers at the everyday objects' side and information consumers at the Internet-based services' side). Only the owners of information channels in the IIP can publish new information to their information channels (*i.e.*, write access), and only information consumers that are listed in an information channel's allowed user list can subscribe to that particular information channel for notifications (*i.e.*, read access). Information consumers can request access to different information channels, and information providers have full rights to add or remove information consumers from/to the allowed user lists of information channels they own. Three services within the healthcare domain have been developed, namely remote health monitoring service, SOS-SMS service and SOS-social media service. This shows how the platform can be utilized to enhance quality of life by means of novel personalized services for patients, in particular, and to society, in general. To avoid a single point of failure, a three-layer deployment architecture of the IIP has been implemented, supporting high availability and scalability by employing redundancy of service components, as well as clustering technology with load balancer. The IIP prototype has been tested to work as intended, and some experiments have been conducted to compare the average total publication and notification times between the standalone IIP deployment and the clustered version, as well as between the inclusion of the security scheme and without it. From the experiments with the current prototypes, it can be concluded that the clustered deployment of the IIP can handle better higher publication rates compared to its standalone counterpart when the number of parameters being used is low. The standalone set-up outperforms the clustered version when the number of parameters increases. In both cases, the incorporation of the security mechanism adds latency overhead.

The HTTP protocol is used by the IIP for message exchanges with both information providers and consumers, due to its pervasive usage on the web. Newer and lighter protocols that are specifically designed for embedded devices, such as the CoAP and the message queue telemetry transport (MQTT), are planned to be supported in the next version of the IIP, especially for interfacing with information providers. JavaScript Object Notation (JSON) will also be supported in the next implementation iteration as an alternative data format to the currently used XML, so that information consumers can choose which data format they prefer for the notifications.

The developed healthcare services described in this article are rather simplistic and straightforward, utilizing only a handful of devices as data sources. On the other hand, the IIP is designed to mediate a wide spectrum of information from a variety of information providers, supporting different application areas. More sophisticated context-aware services that combine information from various different devices, such as home appliances in a smart home environment to assist patients with living independently in their homes, are planned to be developed in the near future.

Optimizations in all three layers of the proposed architecture for deployment are planned to be conducted in the continuation of this work, and further security and privacy enhancements will be investigated and incorporated in the next prototyping round. Additionally, the current IIP prototype is planned to be used in several pilot projects that include real-life patients within the healthcare domain in collaboration with several hospitals and partner companies. In turn, they will provide feedback on how the system could further be improved.

Acknowledgments

This work is primarily sponsored by the University of Agder under PhD research fellowship project grant number 63730.

Conflicts of Interest

The author declares no conflict of interest.

References

1. Ashton, K. That “Internet of Things” thing. *RFiD J.* **2009**, *22*, 97–114.
2. Sarma, S.; Brock, D.L.; Ashton, K. *The Networked Physical World—Proposals for Engineering the Next Generation of Computing, Commerce & Automatic-Identification*; MIT Auto-ID Center, Massachusetts Institute of Technology: Cambridge, MA, USA, 2000.
3. Lake, D.; Rayes, A.; Morrow, M. The Internet of Things. *Internet Protoc. J.* **2012**, *15*, 10–19.
4. Mattern, F.; Floerkemeier, C. From the Internet of Computers to the Internet of Things. In *From Active Data Management to Event-Based Systems and More*; Springer: Berlin, Germany, 2010; pp. 242–259.
5. Gershenfeld, N.; Krikorian, R.; Cohen, D. The Internet of Things. *Sci. Am.* **2004**, *291*, 76–81.
6. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; *et al.* A view of cloud computing. *Commun. ACM* **2010**, *53*, 50–58.
7. Stevens, W.R.; Wright, G.R. *TCP/IP Illustrated: The Implementation*; Addison-Wesley Professional: Boston, MA, USA, 1995; Volume 2.

8. Trossen, D.; Pavel, D. Building a ubiquitous platform for remote sensing using smartphones. In Proceedings of the IEEE the Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, San Diago, CA, USA, 17–21 July 2005; pp. 485–489.
9. Jara, A.J.; Varakliotis, S.; Skarmeta, A.F.; Kirstein, P. Extending the Internet of Things to the future Internet through IPv6 support. *Mob. Inf. Syst.* **2014**, *10*, 3–17.
10. Jara, A.J.; Moreno-Sanchez, P.; Skarmeta, A.F.; Varakliotis, S.; Kirstein, P. IPv6 addressing proxy: Mapping native addressing from legacy technologies and devices to the Internet of Things (IPv6). *Sensors* **2013**, *13*, 6687–6712.
11. Mulligan, G. The 6LoWPAN architecture. In Proceedings of the 4th Workshop on Embedded Networked Sensors, Cork, Ireland, 25–26 June 2007; pp. 78–82.
12. Ludovici, A.; Calveras, A.; Casademont, J. Forwarding techniques for IP fragmented packets in a real 6LoWPAN network. *Sensors* **2011**, *11*, 992–1008.
13. Jara, A.J.; Zamora, M.A.; Skarmeta, A. Glowbal IP: An adaptive and transparent IPv6 integration in the Internet of Things. *Mob. Inf. Syst.* **2012**, *8*, 177–197.
14. Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. Hypertext Transfer Protocol–HTTP/1.1. Available online: http://www.w3.org/Protocols/rfc2616/rfc2616.html (accessed on 31 October 2013).
15. Krishnamurthy, B.; Rexford, J. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*; Addison-Wesley Professional: Boston, USA, 2001.
16. Berners-Lee, T.; Cailliau, R.; Luotonen, A.; Nielsen, H.F.; Secret, A. The world-wide web. *Commun. ACM* **1994**, *37*, 76–82.
17. Curbera, F.; Leymann, F.; Storey, T.; Ferguson, D.; Weerawarana, S. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*; Prentice Hall PTR: Upper Saddle River, NJ, USA, 2005.
18. Fielding, R.T. Architectural Styles and the Design of Network-Based Software Architectures. Ph.D. Thesis, University of California, Irvine, CA, USA, 2000.
19. Jara, A.J.; Zamora-Izquierdo, M.A.; Skarmeta, A.F. Interconnection framework for mHealth and remote monitoring based on the Internet of Things. *IEEE J. Sel. Areas Commun.* **2013**, *31*, 47–65.
20. Jara, A.J.; Zamora, M.A.; Skarmeta, A.F. An Internet of Things-based personal device for diabetes therapy management in ambient assisted living (AAL). *Pers. Ubiquitous Comput.* **2011**, *15*, 431–440.
21. Brenner, M.; Unmehopa, M. The Silo Syndrome and Its Solution. In *The Open Mobile Alliance*; John Wiley & Sons, Ltd: Chichester, UK, 2008; pp. 7–20.
22. Pansiot, J.; Stoyanov, D.; McIlwraith, D.; Lo, B.; Yang, G. Ambient and wearable sensor fusion for activity recognition in healthcare monitoring systems. In Proceedings of 4th International Workshop on Wearable and Implantable Body Sensor Networks (BSN 2007); RWTH Aachen University, Aachen, Germany, 26–28 March 2007; Leonhardt, S., Falck, T., Mhnen, P., Eds.; Springer: Berlin, Germany, 2007; Volume 13, pp. 208–212.

23. Chung, W.Y.; Bhardwaj, S.; Purwar, A.; Lee, D.S.; Myllylae, R. A fusion health monitoring using ECG and accelerometer sensors for elderly persons at home. In Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Lyon, France, 22–26 August 2007; pp. 3818–3821.
24. Fernando, N.; Loke, S.W.; Rahayu, W. Mobile cloud computing: A survey. *Future Gener. Comput. Syst.* **2013**, *29*, 84–106.
25. Huang, D. Mobile cloud computing. *IEEE COMSOC Multimed. Commun. Tech. Comm. (MMTC) E-Lett.* **2011**, *6*, 27–30.
26. Kumar, K.; Lu, Y.H. Cloud computing for mobile users: Can offloading computation save energy? *Computer* **2010**, *43*, 51–56.
27. Eugster, P.T.; Felber, P.A.; Guerraoui, R.; Kermarrec, A.M. The many faces of publish/subscribe. *ACM Comput. Surv. (CSUR)* **2003**, *35*, 114–131.
28. Strom, R.E.; Banavar, G.; Chandra, T.D.; Kaplan, M.A.; Miller, K.; Mukherjee, B.; Sturman, D.C.; Ward, M. Gryphon: An information flow based approach to message brokering. In Proceedings of The Ninth International Symposium on Software Reliability Engineering, Paderborn, Germany, 4–7 November 1998.
29. Pietzuch, P.R.; Bacon, J.M. Hermes: A distributed event-based middleware architecture. In Proceedings of the IEEE 22nd International Conference on Distributed Computing Systems Workshops, Vienna, Austria, 2–5 July 2002; pp. 611–618.
30. Cugola, G.; Di Nitto, E.; Fuggetta, A. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Trans. Softw. Eng.* **2001**, *27*, 827–850.
31. Rowstron, A.; Kermarrec, A.M.; Castro, M.; Druschel, P. SCRIBE: The design of a large-scale event notification infrastructure. *Netw. Group Commun.* **2001**, pp. 30–43.
32. Carzaniga, A.; Rosenblum, D.S.; Wolf, A.L. Achieving scalability and expressiveness in an internet-scale event notification service. In Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, Portland, OR, USA, 16–19 July 2000; pp. 219–227.
33. Huang, Y.; Gannon, D. A comparative study of web services-based event notification specifications. In Proceedings of the IEEE International Conference on Parallel Processing Workshops, Columbus, OH, USA, 14–18 August 2006, pp. 8–15.
34. Huang, Y.; Slominski, A.; Herath, C.; Gannon, D. Ws-messenger: A web services-based messaging system for service-oriented grid computing. In Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid, Singapore, Singapore, 16–19 May 2006; Volume 1, pp. 8–173.
35. Fu, C.; Belqasmi, F.; Glitho, R. RESTful web services for bridging presence service across technologies and domains: An early feasibility prototype. *IEEE Commun. Mag.* **2010**, *48*, 92–100.
36. Kamilaris, A.; Trifa, V.; Guinard, D. Building web-based infrastructures for smart meters. In Proceedings of the First Workshop on Energy Awareness and Conservation through Pervasive Applications, Helsinki, Finland, 17–20 May 2010; pp. 1–6.
37. Heyer, C. *The Å Publish/Subscribe Framework*; Springer: Berlin, Germany, 2009, pp. 99–110.

38. Fitzpatrick, B.; Slatkin, B.; Atkins, M. PubSubHubbub Core 0.3—Working Draft. Project Hosting on Google Code, 2010. Available online: http://pubsubhubbub.googlecode.com/svn/trunk/pubsubhubbub-core-0.3.html (accessed on 31 October 2013).
39. Gregorio, J.; de Hora, B. The Atom Publishing Protocol. Available online: https://tools.ietf.org/html/rfc5023 (accessed on 31 October 2013).
40. Winer, D. RSS 2.0 Specification. Available online: http://cyber.law.harvard.edu/rss/rss.html (accessed on 31 October 2013).
41. Shelby, Z.; Hartke, K.; Bormann, C. Constrained Application Protocol (CoAP). Available online: https://tools.ietf.org/html/draft-ietf-core-coap-18 (accessed on 31 October 2013).
42. Li, S.; Hoebeke, J.; Van den Abeele, F.; Jara, A. Conditional observe in CoAP. Available online: https://tools.ietf.org/html/draft-li-core-conditional-observe-04 (accessed on 31 October 2013).
43. Barry, D.K. *Web Services and Service-Oriented Architecture: The Savvy Manager's Guide*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2003.
44. Michelson, B.M. *Event-Driven Architecture Overview*; Patricia Seybold Group: Boston, MA, USA, 2006.
45. Chappell, D. *Enterprise Service Bus*; O'Reilly Media: Sebastopol, CA, USA, 2004.
46. Trinugroho, Y.B.D.; Gerdes, M.; Mahdavi Amjad, M.M.; Fensli, R.; Reichert, F. A REST-based publish/subscribe platform to support things-to-services communications. In Proceedings of the19th Asia-Pacific Conference on Communications (APCC 2013), Bali Island, Indonesia, 29–31 August 2013; pp. 327–332.
47. Littlewood, B.; Strigini, L. Software reliability and dependability: A roadmap. In Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland, 4–11 June 2000; ACM: New York, NY, USA, 2000; pp. 175–188.
48. Avizienis, A.; Laprie, J.C.; Randell, B. *Fundamental Concepts of Dependability*; University of Newcastle upon Tyne, Computing Science, Newcastle upon Tyne, UK, 2001.
49. Stallings, W.; Brown, L. *Computer Security: Principles and Practice*; Prentice-Hall: Upper Saddle River, NJ, USA, 2008.
50. Nadalin, A.; Kaler, C.; Monzillo, R.; Hallam-Baker, P. Web services security: SOAP message security 1.0 (WS-Security 2004). Available online: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf (accessed on 31 October 2013).