MediaSense as a daemon

Daemonizing a context-aware platform for Internet of Things

Leon Hennings

Kamyar Sajjadi

Department of Computer and Systems Sciences

Degree project 15 HE credits

Degree subject: Computer and Systems Sciences

Degree project at the Bachelor level

2013

Advisor: Jaime Walters

Reviewer:

Swedish title:

Stockholm University

MediaSense as a daemon

Daemonizing a context-aware platform for Internet of Things

Leon Hennings

Kamyar Sajjadi

Abstract

Keywords

Contents

1. Introduction

Around us today is a large assortment of devices whose sensors generate data. This data is mostly used locally or together with other devices in small systems. Smartphones and mobile broadband allow people to have access to the internet wherever they are. As shown in [6], sensors from different devices can be connected to each other using Internet Protocol. This is resulting in an Internet of Connected things, where each thing, whether human or machine can connect and communicate, sharing and digesting information, executing tasks and collaborating to realise massive Immersive Environments of the as described in [32]. This new network is termed the Internet of Things. The Internet of Things, aims to seamlessly fuse people places and things across current communications platforms, realising immersive situations that are enabled through the collection of information from embedded sensors and respond by acting upon corresponding embedded actuators.

Sensors embedded within our environment range from simple sensors such as temperature, humidity, light intensity and occupation sensors, to location and GPS sensors embedded in mobile devices, telephones and automobiles. Applying approaches such as sensor fusion allows us to emulate even higher level sensor information exposing information that is otherwise not directly observable. All these sensors collect data, and with this data developers can build applications that respond to the given data from the sensors. These could range from an application that can automatically regulate the heating in your home to an application that can tells you which way you should take to work according to the traffic on streets.

In order to enable these behaviors and realise the impending immersive paradigm, sensors of a device need to collect and share context information. Earlier research in the area realised the use of middleware systems for this large scale information provisioning. These included both centralized and decentralized approaches. Centralized approaches such as the IP Multimedia subsystem [22] and MQTT [18], exist as web service portals on the Internet, providing a point of connection for entities to provision context information on an Internet of Things. However, these approaches assume the complete availability and reliability systems which are susceptible to DNS errors, denial of service attacks and dynamic IP configuration issues. They are also prone to creating bottlenecks which affect real-time information sharing which means that information accuracy are not guaranteed. This is important for an Internet of Things where real-time is key to creating Immersive Participation Environments. Furthermore, the scalability is limited when using a centralized approach [21]. The problem with scalability and DNS availability therfore makes centralised solutions suboptimal.

In order to fix this it was suggested use distributed approaches. These do not rely on DNS, are more scalable and less prone to DOS attacks. Such approaches include MediaSense [21] and UBIWARE [27] and these have both the *server* and *client* on one machine. Within the mediasense realization, applications are tied to the platform and are started and terminated with the platform. Every application has its own MediaSense instance and communicates with other devices through this instance.

Another thing that needs to be considered when developing middleware for the concept Internet of Things is that devices in our everyday life has limited resources. The current way applications run on MediaSense, where every application needs its own instance of the platform, makes it very inefficient.

1.1 Problem

The Internet of Things aim to enable massive immersive applications. Since the sale of smartphones surpassed that of computers and laptops in 2011 [5] the Internet of Things will heavily incorporate more ubiquitous devices with lower resource availability that can be mass deployed. Therefore, these applications must be able to run on ubiquitous devices.

Distributed Internet of Things middlewares are designed for resource capable devices such as desktop computers and servers. One of the main requirements for Internet of Things middleware defined by Theo Kanter et al. [21] is to have a lightweight middleware. The current design of distributed Internet of Things middleware excludes usage of such ubiquitous devices.

By improving the resource usage of distributed Internet of Things middlewares they will become usable on ubiquitous devices.

1.2 Research Question

Can a distributed Internet of Things platform be redesigned to enable a common service for all context-aware applications running on a device in such a way that it reduces hardware requirements and improves performance.

1.3 Goal

The goal of this project is to make a distributed Internet of Things middleware require few enough resources to be run on ubiquitous devices.

1.4 Scope

The work undertaken during this project is restricted to the MediaSense platform as it is a distributed context-aware platform for the internet of things. MediaSense is written in the Java programming language and our choice of tools to achieve our goals will be based on Java. We will extend the MediaSense platform to support applications written in Java and will only support personal computers and not any mobile platforms like iOS or Android.

2. Background And Theory

Enabling a middleware that supports an Internet of Things requires and understanding of the theories and concepts that support and maintain this connected things paradigm. In this chapter the surrounding concepts that are used for the Internet of Things will be explored.

2.1 Immersive Participation

Immersive participation is focused on participation on the Internet via ubiquitous computing and context-awareness. It enables people, places and things to connect to each other to create Immersive Participation Environments. Immersive Participation Environments provides users with context-awareness everywhere which makes the users participate as if they are in a virtual world with places, things and people in it. Common examples of Immersive Participation Environments today include Google Ingress [26] where users join teams and compete with other teams in a virtual world where they need to take over real world artefacts, TURF [1] where users capture real world places and gain points and RATS Theatre's [33] application called Maryam [9] which is an interactive theater where audio clips is triggered depending on the user's Global Positioning System (GPS) location.

Larger scale immersive applications will benefit from scalable distributed information sharing and also remove bottlenecks and dependencies on centralized web portals on the internet. The way humans interact with each other and things around them will change when sensor information can be shared and accessed ubiquitously. Creating immersive environ that blend the natural world with a seamless internet of things require that we are able to understand the situation of the users in real-time this understanding is termed context awareness.

2.2 Context Awareness

Improving the computers ability to access and understand a user's circumstances give developers more information for building applications that respond and adapt to the user. A way to accomplish this is to not only use data given by the user but also use context information from the users environment. In earlier work Schilit and Thimer [31] is defining context as locations, identities of close people and objects. This definition is too specific. It's not only locations that is interesting as context. According to Dey in [8], a better definition of context is:

> "Context is a combination of any information that can be sensed or received by an entity which is useful to catch events and situations. [8]"

In other words context is information from an entity that gives specific information to increase the understanding of an events environment. An entity can be a person, place or a object that is relevant for the interaction.

Humans implicitly use context information for making everyday decisions. An example of this is when a person is asking another person for directions to a place. The context in this conversation can be the location the persons are standing at and based on this the person giving the directions can either point or give a description of the sequence of right and left turns needed to reach the destination. By having this context information it is possible to give directions to the desired location. This kind of ability is therefore hard to transfer to human interaction with computers. One way of generating and sharing this context information on a large scale is through the use of smart telephones and other ubiquitous computing devices.

One such smartphone is the a Google Nexus 4 [16] which contains, among other things, the sensors an accelerometer to detect acceleration, a GPS to receive location data, a gyroscope to detect rotation, a barometer to detect air pressure and a compass for direction and navigation. By applying sensor fusion [10] other context data can be attained. An example of this would be combining a GPS with a barometer to faster detect altitude. Using these types of data, we can realize context-centric applications. Similar to the earlier example where a person was asking another person for directions to a place, the person can now ask for direction from his phone. The phone will collect context-information from the its GPS sensor and based on this location be able to give the directions. With this extra context information, we can create applications that are context aware. This idea of context awareness is summarized by Dey in [8] as:

> "A system is context-aware if it uses context to provide relevant information and /or services to the user, where relevancy depends on the user's task. [8]"

An application is therefore context-aware if it is able to use context in order to adjust its behavior or the content it is providing. Example of context-awareness in applications is Google Latitude [15]. The application makes it possible for your friends to see your position on a map. Google Latitude uses the locations from a entity (mobile phone or computer) to update your position on the map. Another context-aware application from google is AdSense/AdWords. This application generates advertisement based on the context of the webpage the user have visited. Also Android is a good example of context-awareness in application. When a user rotates a smartphone or a tablet the operating system changes its orientation to landscape mode.

Context awareness can change classical scenarios into intelligent responsive scenarios by using context informations to behave in a special way. Common applications such as home automation can use information to turn on the light at home. In earlier implementations where applications did not use context information and users turn on lamps by

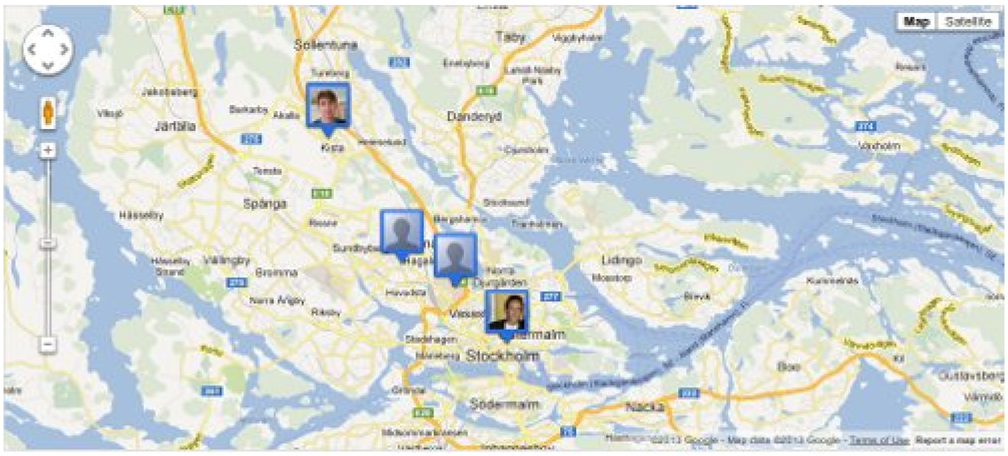
Figure 2.1: A picture of Google Latitude showing contacts shared positions.

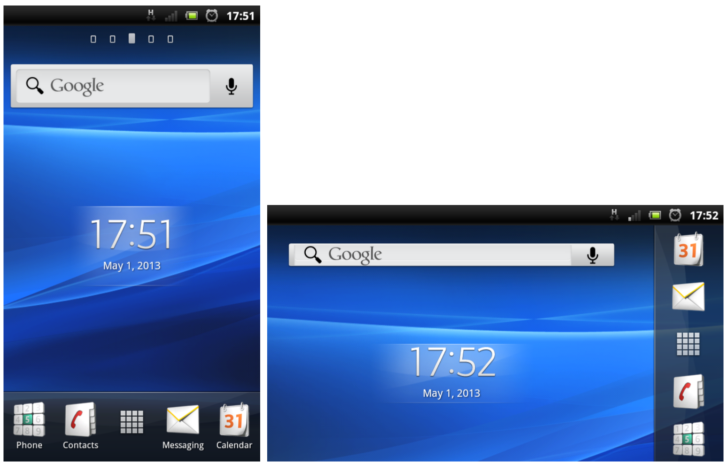

Figure 2.2: Android homescreen on a Sony Ericsson Xperia PLAY rotates based on context information from its sensors when the phone is physically rotated.

manually pressing a button. In comparison, if the developer instead uses the context information the application can interact on information from sensors around the user. For example when a user enters his home the light will be turned on when the application detects his location from the users ubiquitous device. The developer can also use information that provides how the weather is outside, if it's sunny outside the application will act to this and it will not turn on the lights when the user enters his home, but if it's cloudy outside the application will turn on the light. Context awareness on a massive scale is gradually enabled by the advances in pervasive and ubiquitous computing.

2.3 Pervasive & Ubiquitous Computing

Pervasive and Ubiquitous computing describes the philosophy of "everything everywhere computing" and defines the concept of having small computers everywhere. To build context-aware applications contextual data is needed and the ubiquitous computing paradigm can be used to make collection of this data possible. Computers today are everywhere, they can be found in phones, TVs, cars, kitchen machines, watches et cetera. Jens Malmodin et al. [11] predicts that 50 billion devices will be connected together by 2020. As computers became available for personal use it was important to make them easy to use. When computers become pervasive and ubiquitous it becomes important to make the computers themselves smart and easy to use instead. According to Mark Weiser who is the founder of the concept Ubiquitous computing the goals of Ubiquitous computing can be summarized as:

> "Ubiquitous computing has as its goal the enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user. [35]"

With the concept of ubiquitous computing the many computers in our surroundings should always be ready to deliver their services to the user without making the user aware that he is interacting with it. Thus devices become *invisible*, assisting the users without getting in the way or requiring the users to manually interact with them.

According to The Swedish Data Inspection Board [4], ubiquitous computing is a new computer era. In the past one human only had one computer, so called personal computer (PC). In the last years this have been changing. Computers have been integrated into artefacts that humans use in their everyday life. For example if you buy new running shoes you can find a computer chip in them that collects data about your running. Almost everything we use in our everyday life have some kind of computer in it. With ubiquitous computing places and physical objects will be connected to each other and communicate with each other and humans. Because of this the new computer era, ubiquitous computing is growing and one person is using many computers, without knowing that they exist.

If computers will be everywhere, they must be as small as possible. They also need to be cheap and be low-power computers [4]. A example of such computers is Raspberry Pi, a credit card sized computer and can be run with 4 x AA batteries. Other examples is smartphones like Samsung Galaxy 3 and tablets like Nexus 7. According to Moore's Law the coming years devices will be smaller, more powerful and they will be much cheaper [30]. This gives a good chance that computers which already are embedded in all things around us will be powerful enough to run multiple context aware applications. These kinds of devices make it possible for the ubiquitous computing paradigm to grow and smart solutions for collecting context information and share it with others will be needed.

2.4 Applications Using Context Information

To build this big network of context aware applications that interact with users without them knowing it, we need to understand some conditions. Because of the mobility of the devices the communication of context information need to be collected and shared with wireless fidelity (WIFI) or mobile telecommunications technology. If we look at the infrastructure for an Internet of Things platform, it must scale well to increasing amounts of users and must always be available to these users [21].

Another condition to consider is that a large number of applications should be able to run on one ubiquitous device. The ubiquitous devices has limited performance, even if the evolution of the hardware is going forward. Due to their size it makes it important for an application to be lightweight. Ubiquitous computer devices have limited processing capability, small memory space and have limited battery time. The capability of processing is limited which make them not well suited for computation of intensive tasks. A ubiquitous device also have limited amount of available memory. This two conditions makes it more important to have lightweight applications and services on ubiquitous devices. As mentioned battery time is also limited, for example the battery time is decreased when the device is using a lot of network connection. Therefore it is important to make the applications and services efficient in the use of network communication. If we run an context-aware application on a Raspberry Pi we need to consider that the device only have 512 MB ram and that this device should be able to run several instances of similar applications so the footprint from the network communication need to be reduced.

The applications also need near instant access to context-information. Context-information need to be accessed in real-time to provide updated data from sensors. This cannot be provided with a centralized solution [20]. Real-time delivery of context-information between endpoints is important to existing and future mobile applications.

2.5 Sharing The Information

As the network of ubiquitous computers grows bigger, the amount of context information will increase as well and it becomes necessary to understand how this information can be shared. When billions of users and applications is connected to the infrastructure it is important to get the most effective and scalable solution for storing and sharing this kind of data. Applications will need to be able to access the data in real time so the users get the most updated context information.

As an example scenario we could look at a navigation system in a car. For GPS data to be up to date it needs to be updated when the device moves. A car driving on the highway would travel approximately 30 meters per second. For the positioning system to be accurate the position would need to refresh the location at least once a second, 1 Hz [28]. If the location would be shared at the same rate as its refresh rate of 1 Hz it would mean devices would send data every second. This would be a worst case scenario, generally you do not need to update your position if it doesn't change or use a lower refresh rate to compensate for devices moving at a slower speed.

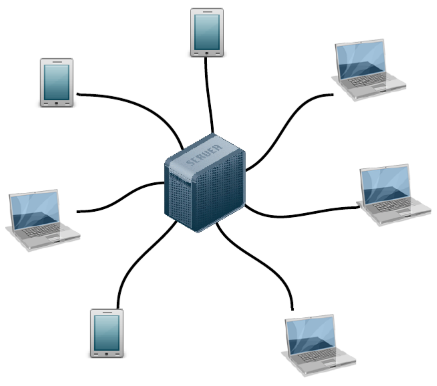

Figure 2.3: Overview of a centralized network

2.5.1 Middleware

To share the information ubiquitous devices need a middleware, a platform that can be run on the ubiquitous devices and enable information sharing. This middleware should be able to provide the functionality for collecting and share context information. The middleware also need to be able to share the context information in real-time. Because of the ubiquitous resource limitations it need to be lightweight and use the resources in an efficient way. This middlewares can share context information in two ways, centralized or decentralized.

2.5.2 Centralized

Centralized middleware solutions are done with a central computer and clients can connect to this computer to store and access context information. In other terms data is hosted and managed in a centralized location, a server, and if a device need to store or access context information it needs to connect to the server. The biggest advantage of the centralized approach is that a server can be updated with new hardware and software to improve its performance. A server can only handle a certain amount of requests per second and as the amount of users and applications increase so will the requests to the server. To handle the larger quantities of requests more servers will be needed to share the load and this will cause the costs to scale with the amount of users. Besides not scaling well the centralized approach is more vulnerable to attacks and crashes because when the server goes down users won't be able to retrieve context information. Some examples of internet of things middleware using this approach are SenseWeb [24] and Xively [37].

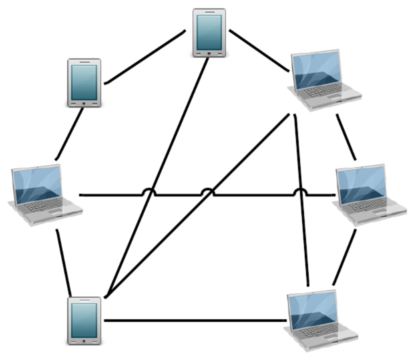

Figure 2.4: Overview of a decentralized network

2.5.3 Decentralized

In a decentralized network the need for a central server is eliminated. Decentralized networks allow computers / nodes to communicate with each other and to share information and resources without using specialized server computers. Each node in the network has both the server features and the client features. All information is distributed over the network of nodes and not only on a centralized computer. Decentralized networks is a technology that is commonly used in file sharing software applications. A example of file sharing application is that is using this technology is Gnutella. Gnutella is an decentralized search protocol that is used to find files. The benefits of decentralized networks is that the decentralized network is more reliable, the dependency to the server is eliminated. If one node is failing this doesn't affect the other nodes in the network. Another benefit of decentralized network is that attacks to the server is eliminated. In a centralized network the server is vulnerable to Denial-of-service attacks or the server can crash and then the whole network gets affected. Also servers don't need to be updated to new hardware when the network is scaling, so the cost of maintaining decentralized networks is less than centralized networks. Ubiware [27] and MediaSense [20] are both decentralized examples of internet of things middleware.

2.6 MediaSense

MediaSense is a middleware platform that supports the concept Internet of Things, it was developed to provide a scalable platform with a real time access to context information [21]. With all sensors around us in our everyday life applications can collect and share context information to each other. MediaSense provides this functionality and enabling support for applications and services to share and collect context information from such as sensors in specific state. The platform is developed so the complexity is low which is enabling users to focus on developing application and services without caring about how context information is shared and how the layers in the platform is interacting with each other [34].

2.6.1 Networking Paradigm

MediaSense is using a decentralized network also called a peer-to-peer overlay to connect nodes in the network. Context information is then persisted in the network and applications can share and collect information in real time from other nodes in the network. A node in the network can act as both a producer and consumer at the same time which enabling bidirectional access to context information. The decentralized network consists of an overlay that use a P-Grid [2] structure as its underlying peer-to-peer technique.

The overlay in MediaSense was first implemented with a Dynamic-Hash-Table. In [34] the authors discovered that the overlay structure needed to be improved and they chose to use a P-Grid structure for the overlay. Because a lot of users with a lot of applications and services is going to share context information it is important to have a overlay structure that is reliable and can handle the scalability [2]. One of the benefits with P-Grid is that it is self organizing. This makes it possible to handle the scalability well.

2.6.2 Applications

MediaSense is written in the programming language Java. MediaSense provide an API that can be used to communicate with the platform. The communication from one platform on a device to another device running the platform is done with messages. The API provides methods for registering new context information and find nodes holding specific context information. Each node attached to the network generates information on a continual basis that is accessed and used by other nodes wishing to do so. In order to do this each node registers itself with a UCI (Universal Context Identifier). The UCI is stored in the distributed network and other nodes can resolve this and get the address where some required information is stored. When a MediaSense instance gets a message a the dissemination layer in MediaSense handles this message and sending it to the application. The dissemination layer is acting like a router and delivers the messages to the right place. Applications have a method that is handling messages that is routed from the dissemination layer.

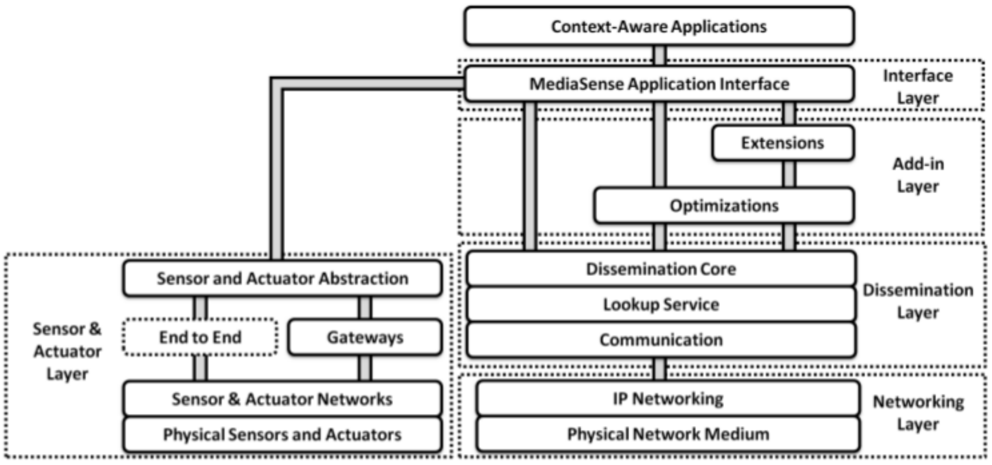

Figure 2.5: MediaSense component architecture

2.6.3 Messages

As mentioned MediaSense communicates with messages. Applications can register what messages they are interested in. When the platform gets a new message from another node in the network the message will be handled by the dissemination core which then will send the message to the application on the platform that is interested in this kind of message. There are several types of messages. The table shows the basic messages that are available for registering and retrieving information.

Message name	Description
REGISTER UCI	Registers a UCI along with the node which is responsible for it.
RESOLVE UCI	Resolves a UCI to the node which is responsible for it.
GET	Fetches the current context value from the node responsible for a UCI. The reply is sent using a NOTIFY.
SET	Changes the current status of an actuator in an end point.
SUBSCRIBE	Makes a subscription request to the node responsible for a UCI, The node then sends a NOTIFY message containing the current context value, either at regular intervals or when the value changes.
NOTIFY	Notifies an interested node of the current context value associated with a specified UCI.
TRANSFER	Requests the manager of a resource to transfer responsibility to another node. This might be full responsibility or partial, where the requester re-creates a copy of the resource permitting improved real time performance.

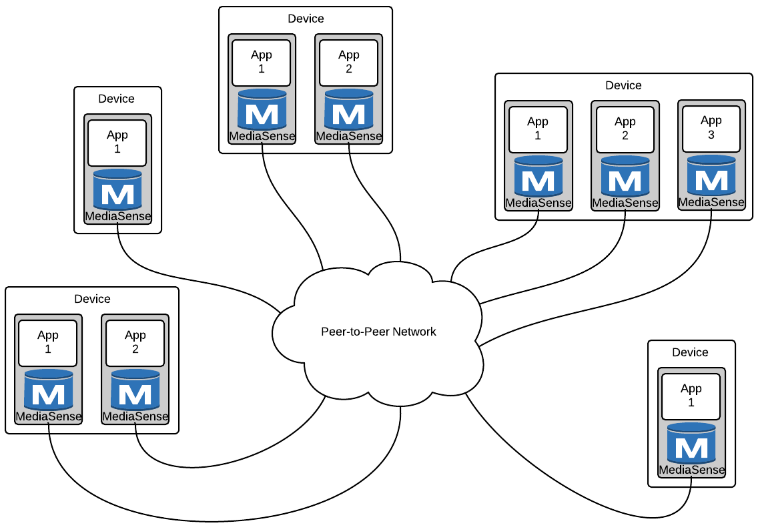

Figure 2.6: Figure showing how the instances of MediaSense is connected to the network

2.6.4 Instances of MediaSense

When an application is run that wishes to use the MediaSense platform, the application must initialize and use its own instance of MediaSense. For example if a user have an application, Application 1 this application need an instance of MediaSense to access context information. If the user have an application, Application 2 running on the same device this application also need its own MediaSense instance to access context information. Every instance of MediaSense is seen as one node, so if we have two instances of the platform on one device this device is acting as two nodes in the network. This is misleading because one node in the network should be one device and not one application.

Each instance of MediaSense requires that a new port be opened so the network layer can communicate with other nodes in the network. This means that if the user is behind a NAT or a firewall the user needs to open up new ports for every instance of MediaSense. The amount of open ports is equal to the amount of applications running on a device. This can be seen as a security issue. With more port open the device that is running the platform is more vulnerable. Not only that the device will be more vulnerable, when a user need several instances of MediaSense and every instance need its own port the network usage will increase. This can affect the battery time of a ubiquitous device in a negative way.

One more thing to consider with an application invoked platform is that the memory usage will increase. Every instance of MediaSense need to use a specific amount of memory. If the platform is running on a ubiquitous device this will limit the number of applications running on the device, because of the several instances of MediaSense.

To make MediaSense more efficient we need to reduce the resource footprint, without losing the functionality. This can be done in several ways. One way is to change the overlay architecture and use a centralized approach for sharing and collecting context information. This solution will make every application connecting to a centralized server or to use a internet portal, for example RESTful API to access the data. The problem with this is that they are centralized and therefore not well scalable and we will lose the functionality of P-Grid. They are also dependent on DNS which means that applications expect that the centralized server are always available. As mentioned before centralized solutions are more vulnerable and can be attacked with Denial-of-service attacks. If the centralized server is having DNS errors the context information can not be accessed and shared to other applications and the applications is usable. Examples of Internet of thing services using this architecture is SenseWeb and Sensei. This two examples is using centralized web-services for sharing context information which makes them having the mentioned issues and are therefore not a good solution for an Internet of things service.

2.7 Inter-Process Communication

Inter-process Communication (IPC) is a term describing communication between two processes through a shared interface. There are a number of variations of Interprocess communication. In operating systems like unix and windows there are a mechanisms for local IPC like files, sockets, pipes, shared memory and semaphores. Inter-process communication using files is simply done by two processes reading and writing to one or more files. Pipes are a way of directing the output of one program to the input of another program. A variation of pipes are named pipes, also called FIFO (first in first out) which are system persistent pipes, often represented as files [23]. Sockets are software abstractions to create a bidirectional channel between processes. They come in two variations, datagram sockets and stream sockets. Datagram sockets are faster than stream sockets but less reliable [23]. Shared memory allows two processes to access the same memory and semaphores are used to signal availability of a resource between processes to avoid information loss, so called race conditions, while writing to the same block of memory from two different processes.

For communication between applications there are Inter-process communication implementations supporting communication with applications running on remote computers through interfaces of a higher abstraction than those for local IPC. These can also be used locally but the abstractions come at a cost in resources.

Remote IPC can be done in two ways unicast and multicast. Unicast is when the communication is sent from one entity and received by another entity. Multicast is when the communication is sent from one entity and received by a group of entities. An example of multicast IPC is message passing. Message passing is asynchronous and as such the message will be handled and replied to when possible. The messages contain data which is used to determine the action or response. Messages are placed in a queue upon arrival and then handled at the recipients convenience. The MediaSense platform uses message passing as a means of communication between nodes over the Internet. An example of unicast Remote IPC is Remote Procedure Calls, RPC. RPCs are unicast and synchronous.

2.7.1 Message passing

Message passing performs IPC by sending and receiving messages. Received messages are placed in a queue and are handled asynchronously. This allows the receiver to prioritize some messages. Message passing provides few abstractions for the developer and requires the data to be marshalled before sending messages and unmarshalled before receiving messages.

The Message Passing Interface, MPI [12], is one such standardized message passing implementation. Sending and receiving operations in MPI can be either blocking or non-blocking. Blocking send and receive operations wait for the application buffer to be free for reuse before returning to prevent race conditions, while non-blocking messages allow the process to proceed as soon as possible.

2.7.2 RPC

Remote Procedure Calls (RPC) are a form of IPC that allow one process to invoke a callable unit [7] located in another logical or physical space. Usage of RPC makes it possible to both interact with programs running on the same physical memory and with applications located within the same network. Remote Procedure Calls are unicast and synchronous which means the sender waits for a response from the recipient before continuing the execution. Remote procedure calls are handled as if the calls were done locally, the calling process waits for a return value before proceeding. See figure 2.7.

Bruce Jay Nelson first defined Remote Procedure Calls as

> "Remote procedure call is the synchronous language-level transfer of control between programs in disjoint address spaces whose primary communication medium is a narrow channel. [25]"

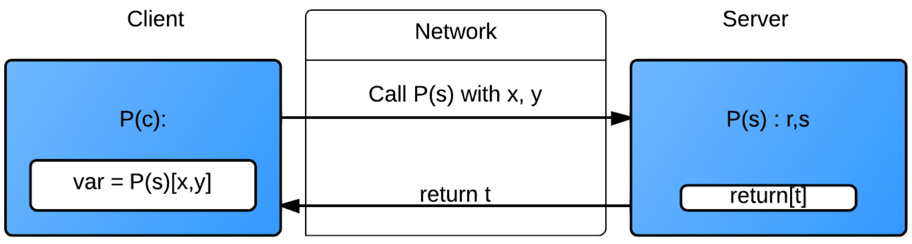

Figure 2.7: Abstract of the events involved in remote procedure calls from a user perspective. The procedure P(c) in the Client calls procedure P(s) located at the Server. The procedure P(s) receive two arguments r and s and return the value t which is stored in variable var back in the client.

When using remote procedure calls, one process is considered the server and one the client. The client is the caller process and the server receive and handle the process and then returns the value. The client and server both have stubs, server-side stubs are called skeletons. These are modules in charge of marshalling the calls, which means packing the parameters of the call in a way that allows them to be stored or transferred via TCP or UDP [29]. The events that occur when a client invokes a remote procedure call start with a call to the client's stub. The client stub receive the parameters from the local call and then marshalls. The marshalled data is then sent to the server by the the client's operating system. The server operating system receives the message and sends it to the skeleton. The skeleton unmarshalls the message to obtain the parameters and invokes the local version of the procedure with the parameters. The server's procedure returns a value which is then sent to the skeleton. The skeleton then in turn marshalls the return value and sends the marshalled data back to the client. The client receive the message, the client stub unmarshalls the return value which then is sent back to the client's procedure [23]. RPC implementations often is language specific. Some implementations, like XML-RPC and JSON-RPC use a common format for describing objects and can therefore be used cross-platform.

2.7.3 RMI

RMI is a Java implementation of RPC with support for Java objects and allows entire objects to be passed and returned as parameters instead of only primitive data types. These objects can be dynamically loaded in the receiving Java Virtual Machine and can therefore be of an object type unknown to the receiver. RMI requires objects to be RemoteObjects, a remote object must implement a remote interface to support remote invocations. When a remote object is sent to another Java Virtual Machine it is sent as a remote stub to the receiving remote. RMI relies on a registry to find other remote objects which have to be registered with a name. Other objects can then do a lookup on the registered name to get a reference to the Remote object. RMI registries can be shared between multiple JVMs on the same machine which facilitates communication from newly created processes to persistent ones.

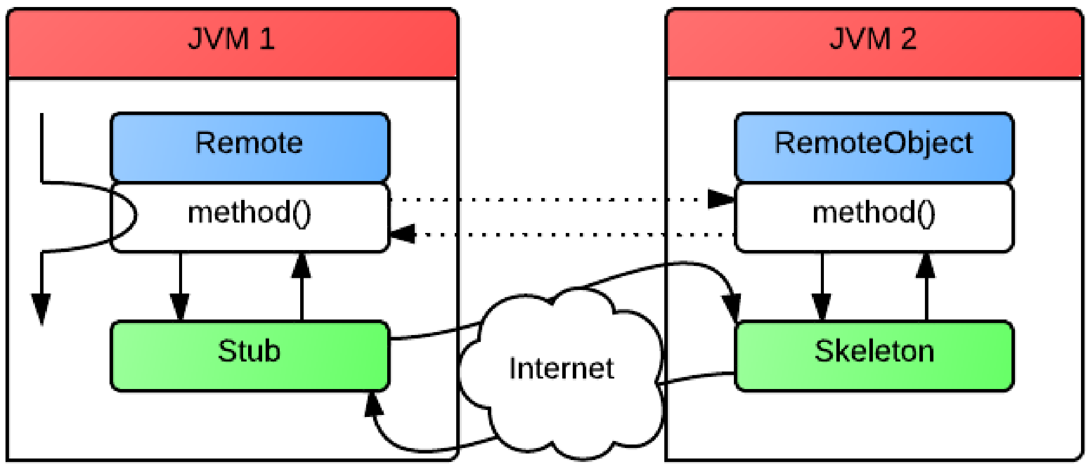

Figure 2.8: The sequence of events in a method invocation with RMI.

2.7.4 CORBA

Common Object Request Broker Architecture is an object-oriented Remote Procedure Call mechanism. Corba is implemented in many different programming languages. This enables softwares written in different languages to communicate with each other. This is done with a language neutral API. CORBA uses an Object Request Broker which provides the mechanism required for distributed objects to communicate with each other. The Object Request Broker determines the location of target object, sends a request to that object and then returns a response back to the calling object. This communication is done over TCP/IP. However, in CORBA it is not possible to bind an Object Request Broker to a specific port. If the client is behind a firewall there is no option to change the port and use another port. This makes Corba firewall unfriendly.

3. Method

To be able to solve the problem it is necessary to rethink and redesign core-components of the an existing middleware. Using traditional quantitative or qualitative research methods would not involve any actual designing. These research methods produce data based on past or present conditions and don't allow researchers to develop artefacts. Design science research differs in one major way from qualitative or quantitative empirical studies. Where empirical studies focus on finding patterns in the past or present of an industry or a field, design science is intended to design and develop new solutions to actual problems [3] and aims to improve and create artefacts that can improve the world [19]. Design science is well suited for this problem because it involves a step where requirements are identified, a design and develop action where the researchers develop an artefact based on the requirements and an evaluation action where said artefact is evaluated to validate the requirements have been met and thus if the artefact solved the problem. Consequently design science is a good method to use for this problem where a redesign of an existing middleware for the Internet of Things is needed.

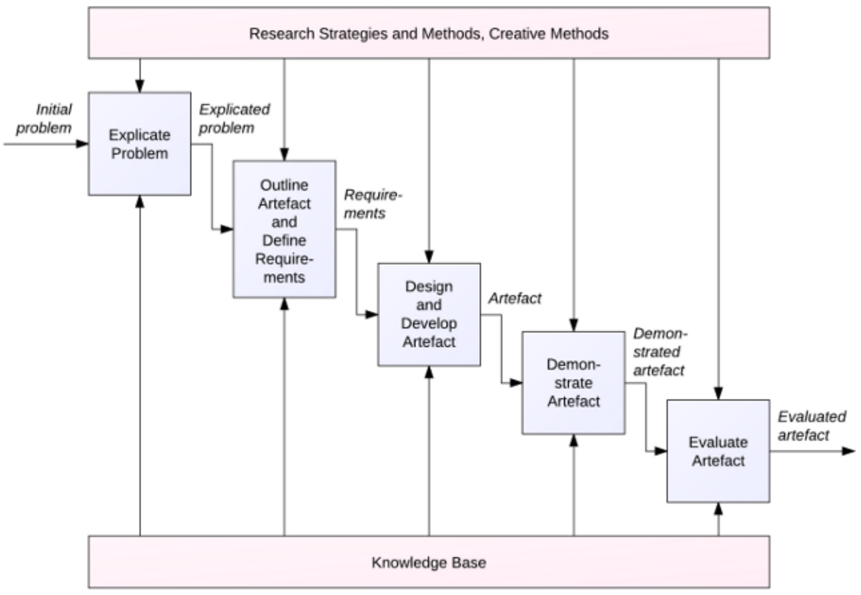

Figure 3.1: Diagram of the Design Science Method [19]

The method chosen for this research is design science as defined by Johannesson and Perjons in A Design science primer [19]. It is well structured and has guidelines that makes it easy to use as opposed to other definitions of design science such as the one laid out in Memorandum on design-oriented information systems research [27] or the one mentioned in Design science in information systems research [17]. Johannesson and Perjons mention that Design Science consists of 5 actions, as shown in figure 3.1. This actions not proposed to be used in a sequence, the steps should be used in an iterative way. In design science it is possible to use other methods to answer questions about artefact. Every action in figure 3.1 should have its own methodology or it can even have several methodologies.

3.1 Research Method For Explicate Problem

To explicate the problem we chose to do a document study of previous publications regarding internet of things middleware. The document study also consisted of reading the source code for MediaSense to get an insight into how MediaSense is constructed and how it worked. Combining this method with interviews of a person with more knowledge about the concept internet of things and knowledge about MediaSense helped to explicate the problem and give a broader knowledge base of the surrounding concepts for internet of things middleware. With a broader knowledge base the problem can then be broken down into a number of subproblems.

An alternative to these methods could be survey using questionnaires, in which predefined written questions can be asked to a respondent. This method can easy be distributed to a large number of respondents. It was not chosen because the researcher cannot ask follow up questions which makes it hard to discuss a problem situation and because there are few persons with knowledge about MediaSense the result of a questionnaire would not explicate the problem, especially when follow up questions is not possible. If there was a larger group of stakeholders for MediaSense a survey could have been used to explicate the problem, but this was not the case and surveys was excluded.

3.2 Research Method For Define Requirements

Due to there only being one stakeholder with sufficient knowledge of MediaSense and as such quantitative methods are not applicable for defining the requirements of the artefact. The qualitative methods we considered were observation, case study, interview and group discussions. To perform an observational study we would need a subject to observe in its natural environment but distributed Internet of Things middleware is still a subject of research and as such the intended environment does not yet exist. Conducting group discussions is not applicable in our situation because there only is one stakeholder. Interviews at first seemed like a good approach because there only is one very involved stakeholder with whom we can conduct the interviews and thus get a deep understanding of the stakeholders needs. A problem with interviews is that the reliability or validity of the answers isn’t guaranteed [14], even though the respondent might answer to their best ability it is possible that not all requirements are immediately unearthed due to the restricted perspectives. Interviews also tend to stifle creativity and are dependent on the questions asked [19] thus resulting in important requirements being missed. Because the stakeholder also was responsible for the artefact the answers given in an interview could be coloured by his own view of the artefact. A case study with both interviews and a deeper study of the artefact was then the best solution as this would make it possible to determine how aspects of the artefact worked before conducting unstructured and open-ended interviews to determine how the solution should work.

3.3 Research For Design And Develop Artefact

With the information gathered in the earlier stages of the design science process architectural changes will need to be designed. For this process participative modelling [19] and document study will be used. The participative modelling will mainly be done by drawing architectural models on a whiteboard and the document studies will be done to research similar solutions and usable technologies.

The development of the design will be done with pair programming [36] which is a practice from the software development methodology extreme programming. According to [36] two programmers working together will find twice as many solutions to a problem than working alone. Also bugs will be found in an earlier state and this will give higher quality on the artefact.

3.4 Research Method For Demonstrate Artefact

A demonstration of the artefact will be done for the stakeholder based on a use case scenario. Because the artefact is a distributed system intended to be used on a large scale and as the artefact still is a research project and the applications intended to be used with it have not been constructed, a real life case cannot be constructed for the demonstration. The scenario will be enacted with a test application and on a smaller scale. The aim of the demonstration will be to cover all requirements and demonstrate that all goals have been met.

3.5 Research Method For Evaluate Artefact

To evaluate the artifact we will need to validate that the requirements have been met. The strategies considered for evaluation are Surveys, Experiments, Case studies, Ethnography, Theoretical Analysis. Doing an ethnographic study would give valuable insights into how an Internet of Things platform would be used and how our redesign would impact usage. Given that Internet of Things still isn't a widely adopted paradigm there would be no precedent to compare cultural impacts to. Such a study would only contribute to the understanding of how people use the Internet of Things and not our artifact specifically. A case study allows for a deep study of the artefact but can be biased by the researchers perceptions, thus doing a case study of an artifact we ourselves developed will be inconclusive as to if the artifact fulfils the requirements. Experiments allow us to set up an artificial scenario similar to that in the demonstration. The experiment will be designed to specifically validate all the requirements. A drawback to using experiments is that the artificial scenario doesn't reflect a real life scenario. To rectify this we will use Theoretical analysis of the results from the experiment. Thus we chose to evaluate the artifact with experiments and theoretical analysis.

3.6 Ethical Considerations

All information uncovered in the interviews will be used confidentially and we will assure the respondent consensually agree to have all answers published in this thesis. The MediaSense platform uses the GNU Lesser General Public License, version 3 [13] and as such, there is no confidentiality we need to observe regarding the source code of it. All test environments used for testing the distributed Internet of Things middleware will be run on a local sandbox for development so the nodes in the network will only contain our own computers.

Bibliography

[1] Andrimon AB. Turf - outdoor addiction. http://www.turfgame.com/. Accessed: 2013-05-12.

[2] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *ACM SIGMOD Record*, 32(3):29–33, 2003.

[3] Ilia Bider, Paul Johannesson, Erik Perjons, and Lena Johansson. Design science in action: Developing a framework for introducing it systems into operational practice. AIS, 2012.

[4] The Swedish Data Inspection Board. Ubiquitous computing - en vision som kan bli verklighet. 2007. Accessed: 2013-05-01.

[5] Canalsys. Smart phones overtake client pcs in 2011, February 2012. Press release 2012/13.

[6] Michael Chui, Markus Löffler, and Roger Roberts. The internet of things. *McKinsey Quarterly*, 2:1–9, 2010.

[7] U.S. Election Assistance Commission. Definitions of words with special meanings. http://www.eac.gov/vvsg/glossary.aspx. Accessed: 2013-05-14.

[8] Anind K Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[9] Dramaten. Maryam. http://www.dramaten.se/Dramaten/Forestallningar/Maryam/. Accessed: 2013-05-12.

[10] Wilfried Elmenreich. Sensor fusion in time-triggered systems, 2002.

[11] Albrecht Fehske, Gerhard Fettweis, Jens Malmodin, and Gergely Biczok. The global footprint of mobile communications: The ecological and economic perspective. *Communications Magazine, IEEE*, 49(8):55–62, 2011.

[12] Message Passing Interface Forum. Mpi: A message-passing interface standard - version 3.0, 2012.

[13] GNU. Gnu project licenses. http://www.gnu.org/licenses/. Accessed: 2013-05-22.

[14] Nahid Golafshani. Understanding reliability and validity in qualitative research. *The qualitative report*, 8(4):597–607, 2003.

[15] Google. Google latitude. www.google.com/latitude. Accessed: 2013-04-29.

[16] Google. Google Nexus 4 tech specs. http://www.google.com/nexus/4/specs/. Accessed: 2013-04-29.

[17] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[18] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. Mqtt-s - a publish/subscribe protocol for wireless sensor networks. In *COMSWARE*, pages 791–798. IEEE, 2008.

[19] P. Johannesson and E. Perjons. *A Design Science Primer.* CreateSpace Independent Publishing Platform, 2012.

[20] T. Kanter, P. Osterberg, J. Walters, V. Kardeby, S. Forsstrom, and S. Pettersson. The mediasense framework. In *Digital Telecommunications, 2009. ICDT '09. Fourth International Conference on*, pages 144–147, 2009.

[21] Theo Kanter, Stefan Forsström, Victor Kardeby, Jamie Walters, Ulf Jennehag, and Patrik Österberg. Mediasense - an internet of things platform for scalable and decentralized context sharing and control. 2012.

[22] Victor Kardeby, Stefan Forsström, Jamie Walters, Patrik Österberg, and Theo Kanter. The updated mediasense framework. In *Proceedings of the 2010 Fifth International Conference on Digital Telecommunications*, ICDT '10, pages 48–51, Washington, DC, USA, 2010. IEEE Computer Society.

[23] Scott M. Lewandowski. Interprocess communication in unix and windows nt. 1997.

[24] Microsoft. senseweb - microsoft research. http://research.microsoft.com/en-us/projects/senseweb/. Accessed: 2013-05-22.

[25] Bruce Jay Nelson. *Remote procedure call.* PhD thesis, Pittsburgh, PA, USA, 1981. AAI8204168.

[26] NianticLabs@Google. Ingress. http://www.ingress.com/. Accessed: 2013-05-12.

[27] Hubert Österle, Joerg Becker, Ulrich Frank, Thomas Hess, Dimitris Karagiannis, Helmut Krcmar, Peter Loos, Peter Mertens, Andreas Oberweis, and Elmar J Sinz. Memorandum on design-oriented information systems research. *European Journal of Information Systems*, 20(1):7–10, 2010.

[28] Matthew D Portas, Jamie A Harley, Christopher A Barnes, and Christopher J Rush. The validity and reliability of 1-hz and 5-hz global positioning systems for linear, multidirectional, and soccer-specific activities. *International Journal of Sports Physiology and Performance*, 5:448–458, 2010.

[29] RFC. Rpc: Remote procedure call protocol specification version 2. http://tools.ietf.org/html/rfc5531. Accessed: 2013-05-22.

[30] R.R. Schaller. Moore's law: past, present and future. *Spectrum, IEEE*, 34(6):52–59, 1997.

[31] Bill N Schilit and Marvin M Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, 1994.

[32] Lu Tan and Neng Wang. Future internet: the internet of things. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 5, pages V5–376. IEEE, 2010.

[33] Rats Teater. Rats teater. http://ratsteater.se/. Accessed: 2013-05-12.

[34] Jamie Walters, Theo Kanter, and Enrico Savioli. A distributed framework for organizing an internet of things. In *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, pages 231–247, 2012.

[35] Mark Weiser. Hot topics-ubiquitous computing. *Computer*, 26(10):71–72, 1993.

[36] Laurie A Williams and Robert R Kessler. All i really need to know about pair programming i learned in kindergarten. *Communications of the ACM*, 43(5):108–114, 2000.

[37] Xively. Public cloud for the internet of things. https://xively.com/. Accessed: 2013-05-22.