Re-Engineering The MediaSense Platform Towards Resource Constraint Devices

A RPC-based Daemon Approach

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: Jamie Walters

Reviewer: Iskra Popova

Swedish title:

Stockholm University

Re-Engineering The MediaSense Platform Towards Resource Constraint Devices

A RPC-based Daemon Approach

Leon Hennings
Kamyar Sajjadi

Abstract

Computers are becoming pervasive in our society, tablets, smartphones, computers embedded in home appliances and televisions. For users to remain capable of handling them all machine to machine communication becomes necessary so the connected devices can create an Internet of Things. The Internet of Things will make use of all collected data now stored separately in devices and allow for a new type of immersive applications by utilizing userdata from several sources. The research project MediaSense is an Internet of Things middleware that allows for communication between devices and distributed sharing of sensor data. MediaSense in its current form is not designed with the trend of smaller, portable computers in mind. To mitigate this a redesign of MediaSense, using the design science methodology, was proposed to make it usable on ubiquitous devices. The proposed redesign intended to make a shared background service, a so called daemon, of the MediaSense platform so functionality of sharing and storing sensor data is shared among the applications using it. To implement this behavior a type of inter-process communication had to be chosen to allow the applications to communicate with the platform. The Java implementation of Remote Procedure Call, called Remote Method Invocation was chosen. The new version of MediaSense was designed and developed according to the requirements outlined by the lead developer of MediaSense. The redesign resulted in lower memory usage and less CPU time. This discovery will make possible the use of Internet of Things middleware and be used everyday and everywhere.

Keywords

Contents

1. Introduction

Smartphones and mobile broadband allow people to have access to the Internet wherever they are. There are many types of devices with sensors which generate data. As shown in [6], sensors from different devices can be connected to each other using Internet Protocol (IP). 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 [39]. 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 physical environment range from simple sensors such as temperature, humidity, light intensity and occupation sensors, to location and Global Positioning System (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 the sensors collect data that developers can use to 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 make this kind of applications possible and realise the impending immersive paradigm, sensors of a device need to collect and share context information [8]. 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 [24] and MQTT [20], 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 Domain Name System (DNS) errors, denial of service attacks and dynamic IP configuration issues. Centralized approaches are also prone to creating bottlenecks which affect real-time information sharing. Without real-time information the freshness and accuracy of the information cannot guaranteed, as mentioned in [43]. 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 [23]. The problem with scalability and DNS availability therefore makes centralised solutions suboptimal.

In order to fix this it is more appropriate to use distributed approaches. These do not rely on DNS, are more scalable and less prone to denial-of-service attack. Distributed

approaches behave like they are both *servers* and *clients*. One such distributed approach to Internet of Things middleware is MediaSense [23] which is an active research project by researchers from Stockholm University and Mittuniversitetet. The server part of MediaSense is responsible for receiving and storing the distributed context data while the client part generates and shares its own context data. 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 not designed for resource constrained devices. The MediaSense platform has been implemented as an Android application but this version of the platform only runs one simple test application [26]. The platform can be run on ubiquitous devices but is not designed to be efficient on such devices. One of the main requirements for Internet of Things middleware defined by Theo Kanter et al. [23] is for it to be lightweight and resource efficient. The current design of distributed Internet of Things middleware causes an resource overhead for each application which in turn excludes usage of multiple applications on such ubiquitous devices.

By improving the resource usage of distributed Internet of Things middlewares they will become usable on ubiquitous devices. Allowing several applications to share necessary resources can reduce the resource overhead.

1.2 Goal

The goal of this project is to reengineer a distributed Internet of Things middleware to require less resources making it able to run multiple applications on ubiquitous devices.

1.3 Research Question

How is resource efficiency impacted by redesigning a distributed Internet of Things middleware to provide a shared service for all context-aware applications running on a device?

1.4 Scope

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 evaluate it and ensure it will be able to run on a device with constrained resources. No cross platform support will be added and mobile platforms like iOS or Android will not be tested.

2. Background And Theory

This research contributes to the Ubiquitous and Pervasive computing paradigm of computer science. In this chapter the surrounding concepts 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 [31] 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 [41] 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 environments 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 [38] 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 [18] 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 get directions from his or her 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 [17]. 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. Another good example of context-awareness in applications exist in all current Smartphone operating systems. 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

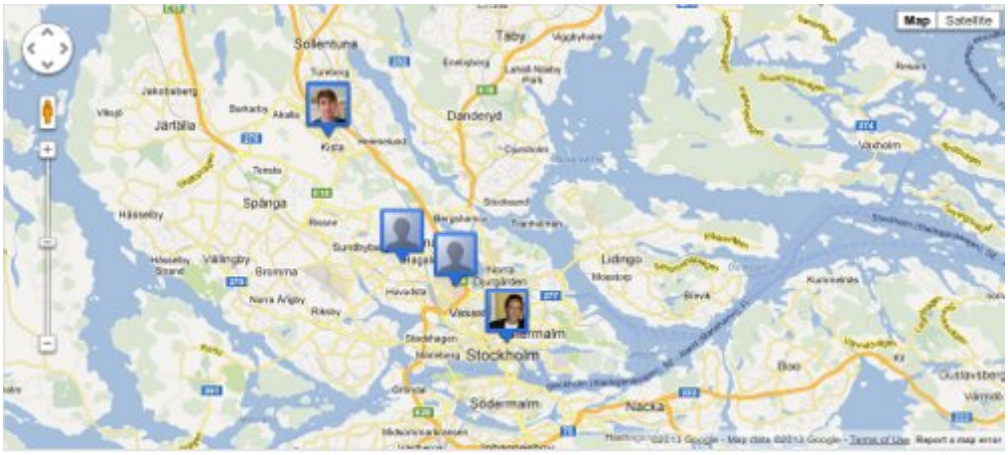


Figure 2.1: A picture of Google Latitude showing contacts shared positions.

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. [45]"

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 [36]. 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]. An 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 [37]. 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 Tendency Towards Resource Constrained Devices

Alan Messer et al [27] have identified a problem with the concept of pervasive computing. Peoples vision is to execute a service on any device without worrying about whether the service is designed for the device. Alan Messer et al believes that it will be difficult to create a service that can be run on all devices, because of the resource constraints on

different devices. Devices processing, memory, network, power capacities differs from each other and services need to be tailored to fit on different devices.

A lot of existing computer software was first developed for computers with high resources. When software runs on a computer with low resources it needs a redesign to fit on the device. The operating system Android is built on Linux which was an operating system that first was used on personal computers. Android Inc redesigned the operating system to fit it on smartphones. The same goes for Microsoft that has developed a version of Windows for tablets and smartphones. Applications like Netflix, Google Chrome and Utorrent have been redesigned to fit different devices. Software can sometimes be run on a device for which it wasn't designed to run on, but the software is not tailored for those devices and will therefore be less efficient causing reduced performance.

2.5 Applications Using Context Information

To build this big network of context aware applications that interact with users without them knowing it, some conditions must be understood. Because of the mobility of the devices the context information need to be collected and shared through wireless network. Moreover, the infrastructure for an Internet of Things platform must scale well to increasing amounts of users and must always be available to these users [23].

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 a context-aware application on a Raspberry Pi we need to consider that the device only has 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 [22]. Real-time delivery of context-information between endpoints is important to existing and future mobile applications.

2.6 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 [33]. If the location would be shared at the same rate as its refresh rate of 1Hz 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.

2.6.1 Middleware

To share the information ubiquitous devices need a middleware, a platform that can 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.

2.6.2 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 [28] and Xively [47].

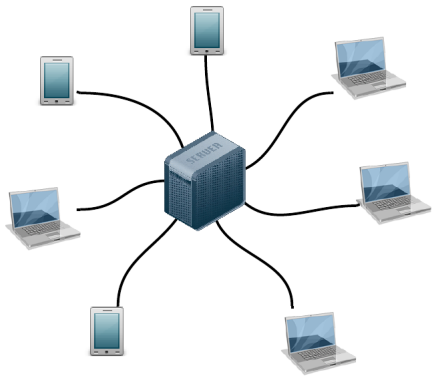


Figure 2.3: Overview of a centralized network

2.6.3 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 [32] and MediaSense [22] are both decentralized examples of internet of things middleware.

2.7 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 [23]. 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

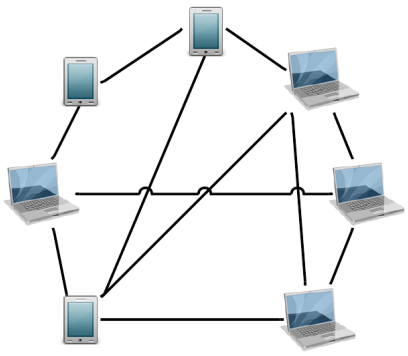


Figure 2.4: Overview of a decentralized network

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 [44].

2.7.1 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 [44] 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.

2.7.2 Applications

MediaSense is written in the programming language Java. MediaSense provides 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

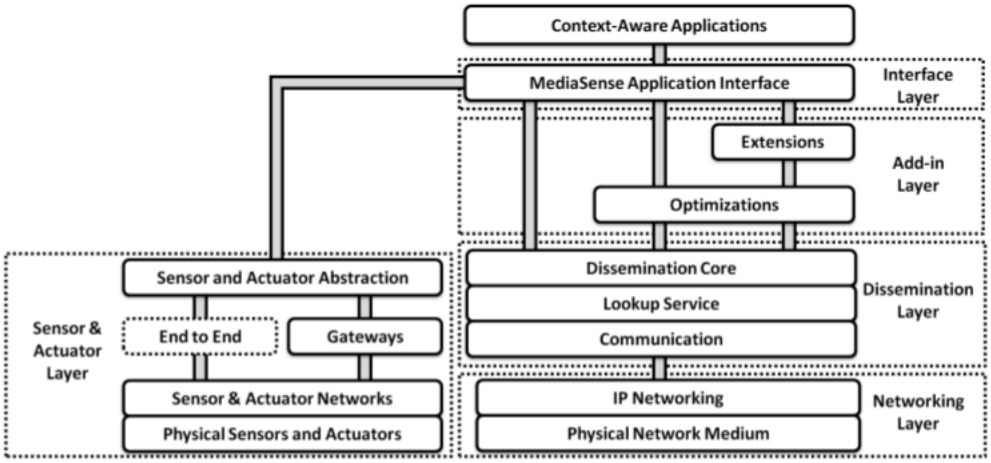


Figure 2.5: MediaSense component architecture [23]

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.

2.7.3 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.

2.7.4 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

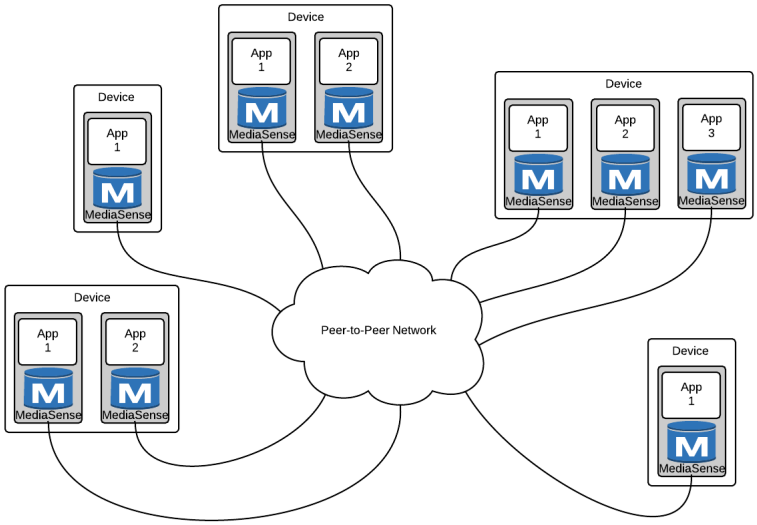


Figure 2.6: Figure showing how the instances of MediaSense is connected to the network

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.

2.8 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 [25]. 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 [25]. 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.

2.8.1 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.

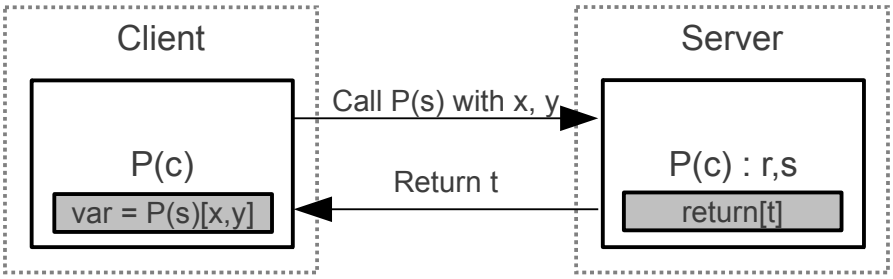


Figure 2.7: Abstract of the events involved in remote procedure calls from a user perspective. The procedure P(c) in the Client calls procedure P(s) located at the Server. The procedure P(s) receive two arguments r and s and return the value t which is stored in variable var back in the client.

2.8.2 Remote Procedure Calls

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. [30]"

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 [34]. 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 [25]. 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.

2.8.3 Remote Method Invocation

Remote method invocation (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.

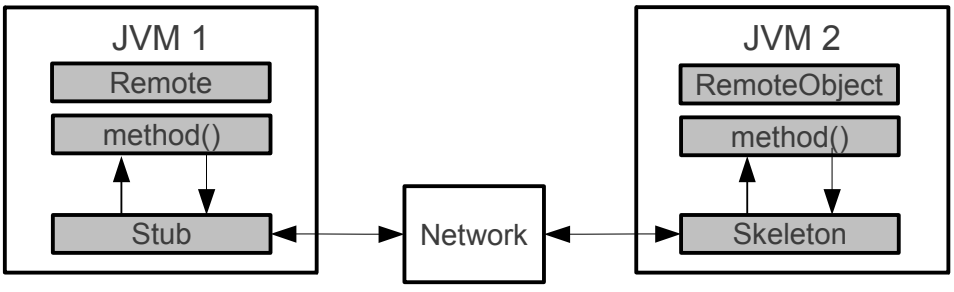


Figure 2.8: The sequence of events in a method invocation with RMI.

2.8.4 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 [21]. Design science is well suited for this problem because it involves a step where requirements are identified and a design and develop action where the researchers develop an artefact based on the requirements. Lastly there is an evaluation action where the artefact is evaluated based on the requirements to see if they 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.

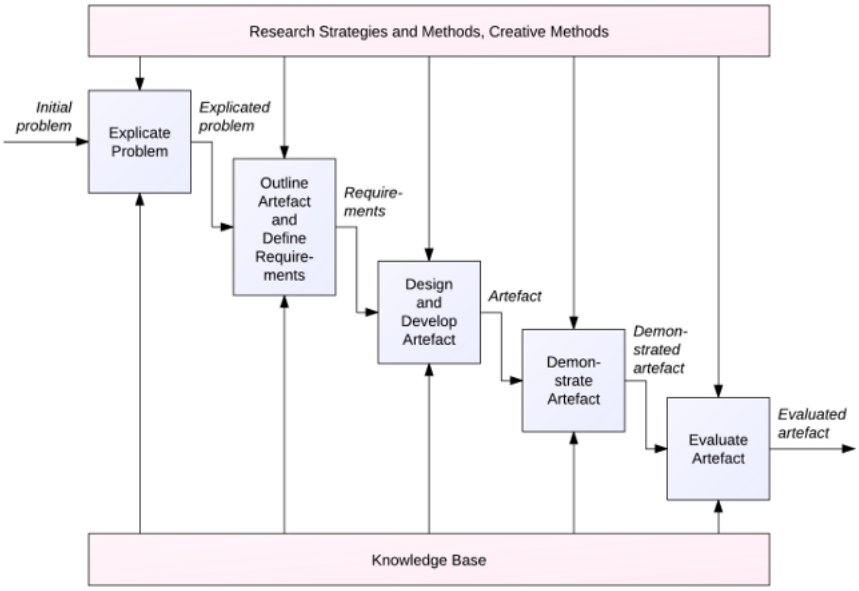


Figure 3.1: Diagram of the Design Science Method [21]

The method chosen for this research is design science as defined by Johannesson and Perjons in A Design science primer [21]. 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 [32] or the one mentioned in Design science in information systems research [19]. Johannesson and Perjons mention that Design Science consists of 5 actions, as shown in figure 6.1. 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 6.1 should have its own methodology or it can even have several methodologies. MediaSense is still a research project and real life use cases would require applications and therefore the demonstration action will not be performed as it's not valuable to this research.

3.1 Research Methods

3.1.1 Explicating The Problem

To explicate the problem a case study of MediaSense is done. The research methods for this case study is 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 be broken down into a number of subproblems.

An alternative to the research strategy case study 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 strategy like survey could have been used to explicate the problem, but this was not the case and surveys were excluded.

Also action research was thought about to use for explicating the problem. Action research was not used though the researchers does not have any experience of the problem and can therefore not offer fresh perspectives to the problem to explicate it. The lead developer of MediaSense have a broad knowledge about the problem and interview can be done when a case study of existing middleware is done. After collecting data for the explicated problem an analysis is done by discussing the problems the researchers found in the source code with the lead developer to confirm the explicated problem.

3.1.2 Defining Requirements

The research strategies considered were surveys, case study and action research. Surveys was discussed to used as research strategy for defining requirements. Surveys is good to

use when researchers want to investigate the needs and wants of many stakeholders [21]. MediaSense is a research project still in progress and because of this there are few people with knowledge of how it works. This means there are not many stakeholders to send the surveys to and therefore this strategy was not used. Because of this quantitative research strategies are not applicable for defining the requirements of the artefact. Looking instead at qualitative research strategies, action research was first considered as a research strategy for defining requirements. Action research is a research strategy which is good to use when the researchers knowledge exceeds that of the stakeholders and the stakeholder have limited understanding of the new artefact [21]. This is not the case for this project since the lead developer has broad knowledge about the problem and a good understanding of the new artefact that is going to be developed. Therefore, Action research is not chosen for this action in design science. A case study as research strategy with both interviews and a deeper study of the existing artefact was 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. A problem with only using interviews is that the reliability or validity of the answers isn't guaranteed [16], 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 [21] thus resulting in important requirements being missed. Because the lead developer also was responsible for the artefact the answers given in an interview could be coloured by his own view of the artefact. Therefore a deeper study of the existing artefact was done to complete the interviews. The study will involve both reading the code and benchmarking the software.

Alternative research methods observation study and group discussion were considered. To perform an observational study the researchers 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 is only one lead developer that can participate. Interviews at first seemed like a good approach because there only is one very involved developer whom we can conduct the interviews and thus get a deep understanding of the lead developers needs.

3.1.3 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 [21] 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 [46] which is a practice from the software development methodology extreme programming. According to [46] two programmers working together will find twice as many solutions to a problem compared to working alone. Also bugs will be found in an earlier state and this will give higher quality on the artefact. To keep the timeline and give updates to the lead developer of MediaSense weekly meetings was arranged where the progress of designing and developing the artefact was presented.

3.1.4 Evaluate Artefact

To evaluate the artifact it is necessary to validate that the requirements gathered in the earlier action *defining 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.

3.2 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 [15] 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.

4. Explicate Problem

The explicate problem step in Design Science [21] is to formulate precisely the initial problem and investigate its underlying causes. To define the problem as precisely as possible a scenario will be defined. The problem was split into several sub-problems. As mentioned before the research method for explicating the problem is a document study of previous publications regarding distributed Internet of Things middleware, document study of reading the source code for MediaSense and interviews with the lead developer of MediaSense.

4.0.1 Research Question

As the authors discuss in [21] a research question was formulated to form the basis of the problem explication. For the problem explication part of the project the research questions is the following:

> "What are the major issues with current implementations of distributed Internet of Things middleware which make them unable to realize the Internet of Things vision?"

4.0.2 Method Application

To explicate the problem a document study of previous publications regarding distributed Internet of Things and middleware for it was done. Some of the publications regarding MediaSense were provided by the stakeholder [22], [23], [44]. These provide background on Internet of Things, what MediaSense is and how it works theoretically. Publications about Internet of Things and the surrounding theories were found by searching IEEE Xplore, ACM Digital library and Google Scholar. The main search queries used were *Internet of Things*, *Internet of Things middleware*, *Internet of Things centralized* and *Internet of Things decentralized*. Because the concept Internet of Things is a popular research area, a lot of articles was found. Publications was selected by reading the abstract to see if the publication were able to expand the knowledge base for the problem, searches for keywords were done and ranking publications after number of citations was done to narrow down the search result.

After getting a grasp of the current state of Internet of Things middleware unstructured interviews were conducted with the lead developer of the MediaSense platform. This respondent has a lot of experience with distributed computing and a good understanding of how Internet of Things middlewares works, therefore this respondent was the best person to interview. The interviews were conducted in an informal manner in the respondent's office and in a conference room with access to a whiteboard. No recording or notes were

done for the interviews thus availed both authors to be active in the interviews and discussions could be done without thinking of recording or taking notes. These interviews were conducted iteratively and in parallel with a document study of the existing source code of MediaSense. This document study yielded questions for upcoming interviews and the interviews in turn gave more information about the inner workings of Internet of Things middlewares and how to proceed the study of MediaSense's source code.

Interviews began with trying to discover the problematics of the chosen Internet of Things middleware. The lead developer of MediaSense first presented a problem that had been identified with MediaSense, a large resource overhead when running multiple applications. The first set of interview questions served to fill the gaps that documentation normally would. The questions aimed to explore how MediaSense worked and to get a broader knowledge base. After studying the source code and getting an understanding of the architecture, the interview questions explored certain modules functionality and how they worked. When the interviewee responded to the questions in an uncertain manner, the follow up questions were formulated to ascertain how they were supposed to work or how it was intended they should work.

After the application of methods used to explicate the problem an analysis was done on the result from the interviews and the document study. This was done by putting together the result from the interviews and the knowledge that was built when the document study of code was done. By using the answers from the interviews and reproducing the problems mentioned by the respondent on a developer instance of MediaSense, the problem was clearer from a technical view.

4.0.3 Results And Analysis

The following scenario describes a use case scenario of an Internet of Things platform.

Scenario

Johan is CEO for a company in Stockholm. He is always on the move from meetings with his co-workers at work and to his daughter's football practice after work. His apartment is equipped with broadband and he has a Raspberry Pi computer connected to the Internet through a router. This Raspberry Pi is working as an information hub for Johan. The computer is running a distributed Internet of Things middleware and has 20 application installed. Johan has bought a new temperature regulator for his apartment. This regulator comes with an application that can be installed on his information hub. The temperature regulator is context-aware, it gets GPS information from Johans mobile phone and regulates the temperature in his apartment according to this. When he leaves home every morning the heating in the apartment turns down. At work Johan receive notifications to leave for meetings based on where the meetings are and how long it will take him to get there. When Johan leaves work his car's navigation system checks with his calendar if

he has anything on his schedule, discovers his daughter has football practice and plots a course to the practice to pick her up. Johan's mobile phone alerts his heating regulator as he approaches his home and the heating is turned back on. Another application is connected to a thermometer by Johan's house and collects information about the temperature outdoors and adjusts the heating of according to this information.

Define problem

Distributed Internet of Things middlewares are resource heavy, which makes them inefficient to run on ubiquitous devices. Centralized Internet of Things middleware can solve this problem, but centralized approaches are more vulnerable and it can not be guaranteed that the server is always up and ready to respond to clients requests. Therefore, a distributed Internet of Things middleware needs to be redesigned to reduce the resource footprint. The chosen middleware for redesign is MediaSense. MediaSense is being developed by researchers at the Department of Computer and System Sciences at Stockholm University and is open source.

After the interviews and document study of the source code of MediaSense the main reason for the resource overhead was identified. MediaSense in its present form makes it necessary to run the platform once for every application. Every application running on a device needs its own instance of the middleware. This makes it necessary to run a middleware for every application running on a device, which is the main reason of the resource overhead.

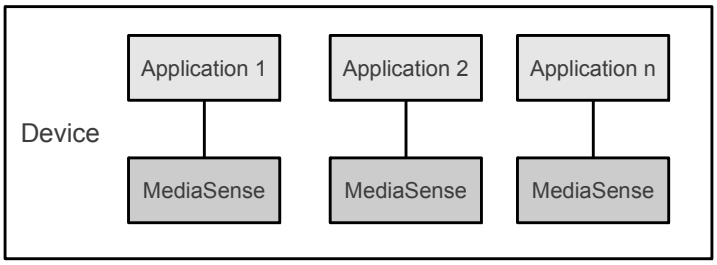


Figure 4.1: The current state of the Mediasense platform showing how every application need its own instance of MediaSense.

A sub-problem to the multiple middleware instances is that every instance need its own port for communicating. Because MediaSense is communicating over IP every instance needs its own port open. This means that a user needs to open new ports on the router and firewall for every application running on the device. When several instances of MediaSense is running on a device the network traffic increases, more processing power is used and more memory is used.

Motivation

To make the scenario presented above possible an Internet of Things middleware it is needed to facilitate the communication between devices. Because the devices used in the scenario are mobile this middleware needs to be resource efficient. The devices need to handle a multitude of applications. Given the current state of MediaSense, the scenario will not be possible because of the resource overhead. To make the Internet of Things concept possible on ubiquitous devices this resource overhead must be dealt with.

Analysis Of Problem

With distributed Internet of Things middlewares every instance of the middleware is both a *server* and a *client*. Every instance of the middleware have its own network layer and database layer for storing context information. In a centralized middleware the server functionality can be moved to a centralized computer and therefore centralized middleware are more lightweight. This is one of the reasons distributed Internet of Things middleware are resource heavy.

To reduce the resource overhead it is prefered to change the architecture of MediaSense. Changing the architecture so that only one shared instance of MediaSense is running can reduce the resource used on a device. As shown in figure the desired architecture of MediaSense MediaSense will be run as an underlying *daemon* and every application needing the services from the platform can use the daemon. This also solves the subproblem with multiple network layers on one device. Devices only need one open port on router or firewall to communicate with other nodes in the network.

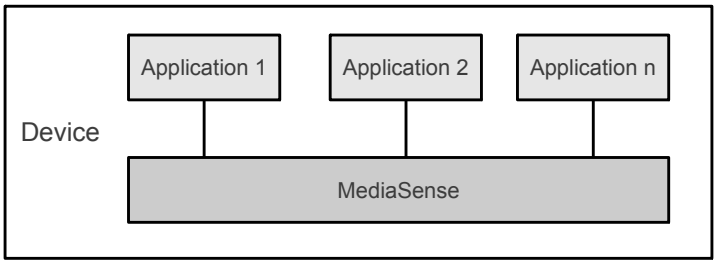


Figure 4.2: The desired state of the Mediasense platform.

5. Outline Artefact and Define Requirements

This activity in design science involves identifying and outline an artefact that can address the explicated problem and also define requirements for the artefact [21]. The requirements will help to get required functionality and constraints for the new artefact.

5.0.4 Research Question

> "What requirements are needed and important to the stakeholders for a new Internet of Things middleware?"

5.0.5 Method Application

The case study of MediaSense began by examining its architecture. The problem explication made it clear that the platform needed to be run as one separate instance and give applications an interface to communicate with it through. This required knowledge of what parts of MediaSense the applications had access to and deciding which parts of MediaSense should be shared between applications. Interviews with a stakeholder, the lead developer of MediaSense, was conducted to determine the requirements for the artefact. The interviews were connected to the case study where investigations of different scenarios were done. These were done in the form of informal open-ended interviews with the goal of understanding which the problems were with the old artefact and how the new artefact should work. From the information collected from the case study and the open-ended discussions a list of requirements was extrapolated. This activity was done in an iterative way under the design and develop activity as new issues were detected. These issues were brought to the stakeholders attention and discussed in order to define new requirements. The MediaSense platform was profiled to obtain a benchmark for the memory consumption before our redesign.

5.0.6 Outline Artefact

To solve the problem a new type of Internet of Things middleware is required to be developed. Instead of developing a completely new middleware an existing one was chosen to be redesigned. The analysis of the problem explication uncovered that the main cause for the resource overhead in MediaSense is the distributed approach of the network layer. This is because every application has its own middleware and the middleware has its own server and client to communicate with other nodes. A redesign is necessary to build a middleware with one common instance of a middleware for every application. The new

artefact is a fork of the existing middleware MediaSense. In [21] four types of artefact is defined: *constructs*, *models*, *methods* and *instantiations*.

> "**Constructs** are terms, notations, definitions, and concepts that are needed for formulating problems and their possible solutions. **Models** are used to depict or represent other objects. **Methods** express prescriptive knowledge by defining guidelines and processes for how to solve problems and achieve goals. **Instantiations** are working systems that can be used in a practice."

The artefact type will therefore be an instantiation.

5.0.7 Results And Analysis

This section shows the requirements found from the case study and interviews. Requirements are categorized in functional requirements and non-functional requirements [35]. The requirements all pertain to the properties mentioned in [21].

Functional Requirements:

Several applications

One instance of the middleware should be able to handle several applications. This is requirement has the property modularity for allowing any combination of applications to use the platform simultaneously.

Interface to applications

Applications should be able to communicate with the middleware through an interface. This is a requirement with the properties flexibility and maintainability allowing the middleware to be changed without destroying compatibility with applications.

Platform as daemon

When several applications are running on one device one shared instance of MediaSense should be used for the applications. This can reduce the resource overhead and help solving the underlying problem with a resource heavy middleware. This requirement has an Interoperability property, which means that the artefact has the ability to work together with other artefacts [21].

Common network layer

The case study showed that a lot of messages was sent from a platform to other platforms. If a common network layer could be used for all applications running on one device the network usages would decrease. With less network usage the battery of ubiquitous computers will have better battery time. This requirement has both the property of being efficient and modular.

Application independent

Applications should be able to start and stop independently of the platform. A crashed application should not affect the execution of the middleware itself. This requirement has the property robustness which means that it have the ability to cope with failures, errors and other problems during execution [21].

Gateway

Run MediaSense platform on a gateway and applications on connected ubiquitous devices. This requirement has the property accessibility because it allows for a greater variety of devices to use MediaSense. This requirement also has the property of efficiency because the connected ubiquitous devices will only need to run the applications.

Messages with scope

Messages to other MediaSense nodes should have a scope, either for a specific application or to all applications on the node. This requirement was added to maintain the coherence property of MediaSense. With several applications running on a node messages must be able to be sent to the specific applications.

Non-functional requirements:

Less Memory Usage

The redesign of MediaSense should use less memory so it is able to run multiple applications on ubiquitous devices. This requirement has the efficiency property.

Java Version

Because MediaSense was written using Java 1.5 the stakeholder would prefer if the redesign were done using the same version of java. This requirement makes the maintenance of the new artefact easy for the stakeholder.

Object Oriented Style

The same object oriented style which had been used to write MediaSense should be adhered to. This requirement has the properties maintainability and elegance. When using objects as parameters for method calls only a few parameters need to be send, the objects can hold a lot of data that need to be used in the method. This is the code style the stakeholder prefers.

Unmodified Overlay

The stakeholder preferred to not change the network layer of the old artefact. If possible, the network overlay module should be left unmodified. This requirement is to uphold the maintainability property.

6. Design and Develop Artefact

This activity aims to design and develop the artefact that is going to solve the problem announced in the explicated problem. The artefact is based on the requirements gathered in the previous activity. This activity involves some document study and modelling to find the best design for the artefact.

6.1 Development Process

The development process was done using Agile Software Development Techniques, especially Scrums daily meetings and Pair Programming. The requirements were split into smaller tasks that were added in a project management tool. When new functionality was detected these functions were added to the feature backlog. When functions was implemented new functions from the backlog was picked out and developed. One day every week a meeting was arranged with the stakeholder to give update on the development progress. In these meetings problems was highlighted and sometimes new requirements were announced by the stakeholder. With the requirements collected from the stakeholder it was clear that the chosen middleware, the MediaSense platform, needed to be split into two pieces. The resource heavy modules in the middleware should be able to run once on a device and several application should be able to use them.

The way MediaSense worked meant an application was started together with the platform behaved as a single instance. Thus, the application had access to all of the platforms functionality. Splitting up the application and platform would mean the application would run as a separate instance and therefore not have access to the platforms methods, a means of communication would be needed between application and platform to make this functionality available. MediaSense is written in Java and Java programs are executed in Java Virtual Machines (JVM). This means that the platform would run in one JVM and then several applications can be started and communicate with the platform from their own JVMs. Because one of the requirements previously discovered was that applications should be able to run on other devices and use the platform as a gateway, this communication would need to work remotely over network.

A document study of Remote Procedure Call (RPC) implementations was initiated to decide how applications should communicate with the platform. By comparing the gathered information about different RPC implementations we decided on using Remote Method Invocation which is a Java specific RPC implementation. RMI supports sending entire Java objects as parameters which complies with the requirement of keeping the Object oriented style of MediaSense. It also forces all remote objects to throw RemoteExceptions which facilitates the Application Independent requirement because the platform

can catch the exceptions thrown by applications and avoid going down with them. The main drawback with using RMI is that it is not as lightweight as other RPC implementations, but the resource overhead this causes is small in comparison to running several instances of the MediaSense platform.

To find the best way to change the architecture of MediaSense several participative modelling sessions were held where models of the old architecture were drawn on the whiteboard and changes was applied on these models to find the best solution to solve the problem.

6.2 Artefact Description

6.2.1 Network Overlay

The network overlay handles the communication with other nodes and stores peer data in a database. The overlay has remained the same as in the old version of MediaSense and thus satisfies the requirement that the overlay should remain unmodified.

6.2.2 MediaSense Messages

MediaSense communicates with other nodes in the network by sending and receiving messages. Messages have scope and can either be sent to a specific application or to all applications at a peer node. To create an application scoped message the application ID is sent as an argument to the constructor and the messages are then sent to the application that is identified with this ID. Messages can also be sent as peer messages, the message is sent to all applications on the receiving peer-node. In the original version of MediaSense, because every instance of the platform only ran one application, the messages had no scope and thus the applications would receive all messages. In the version that has been developed in this project the dissemination core can redirect messages to specific applications. With the possibility to send messages to a specific applications the new version of MediaSense fulfills the requirement that applications should have scope.

6.2.3 Dissemination Core

The Dissemination layer in MediaSense includes different components, dissemination core and lookup service. The lookup service finds and resolves other entities who connects to the network. The dissemination core is working as a router for the messages. When the platform is receiving a messages the dissemination core handles this messages and sends it to the applications that is interested of this message. If an application ID is specified the message will be sent to the application with this ID, if the messages type is set to be a peer message the message will be sent to all application connected to the platform. The dissemination core and the use of RMI satisfies the requirement that the artefact should handle several applications.

6.2.4 MediaSense Platform Interface Layer

The MediaSense Platform Interface Layer initiates the core components of the MediaSense platform, the Dissemination Core, the Pgrid lookupservice, and Pgirds module for network communication. In the old version it was also used to expose the functionality of the MediaSense Platform to the applications. In the new version this component is only used by the RMI server. The client applications now access this functionality through the RMI Proxy which in turn calls the Interface Layer. This satisfies the requirement that applications should have an interface to the platform and that they should use a common network overlay.

6.2.5 MediaSense Platform RMI Proxy

This component is a RMI server that register itself to the RMI registry and makes it possible for RMI client to connect to it. The RMI proxy provides methods so the applications can communicate with the RMI server which is calling methods in the provided MediaSense Platform Interface Layer. The RMI server is acting as a shaded API so applications can call methods that is provided by MediaSense. This component makes it possible for several applications to connect to it and therefore only one MediaSense instance is needed for communication with the platform. This allows MediaSense to be run as a background process and thus satisfies the requirement that MediaSense should be able to run as a daemon.

6.2.6 MediaSense Application Interface

The application interface is used when a developer is developing an application. The developer extends this interface to get access to functionality that is needed for applications running on MediaSense. To create a MediaSense application few things are required.

- An application extending the interface must define its own unique application ID. This ID can be used to set the scope of a message to a specific application.
- Applications must register what types of messages they are interested in receiving using the RMI Proxy's registerListener method.
- An application need to register itself on the platform by sending a reference of itself to be stored in the platforms list of applications. This is done by providing the applications ID as a parameter to the method called registerApplication in MediaSense Platform RMI proxy.
- The application interface contains one method that developers need to override called handleMessage. This method is used for handling incoming messages to the application and is responsible for responding to these messages.

To communicate with the MediaSense platform an application must first know the IP address of the device where the RMI registry is located. If the platform runs on the same

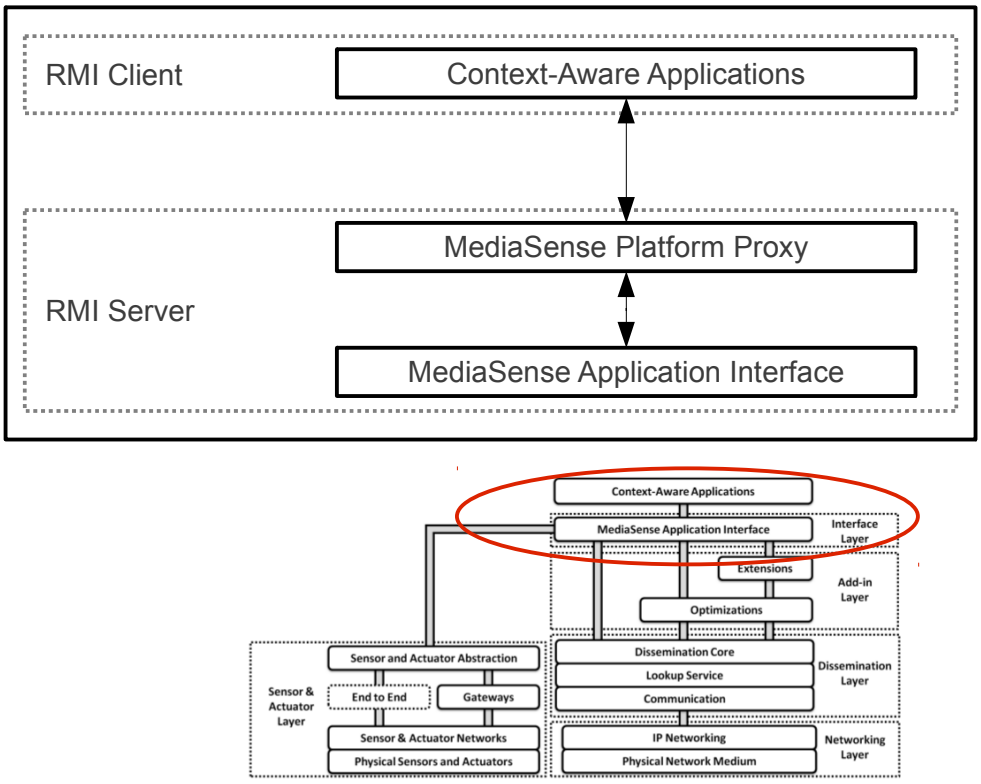


Figure 6.1: Figure showing how MediaSense Platfrom Proxy and application communicate with each other and where the RMI client and server is located in the architecture.

device as the applications an arbitrary port is used. After connecting to the registry the RMI Proxy is located through a lookup on the registered name *mediasense* and an instance of it is saved in the application. When the application interface needs to communicate with the platform a method called getPlatformInterface is available which is returns the instance of the RMI Proxy on which all calls to the platform then can be done.

7. Evaluation

This activity is for evaluating the artefact, addressing both the defined requirements and the explicated problem. This activity shows if the designed and developed artefact solves the problem and shows how the evaluation of the artefact was done.

7.1 Research Question

> "How has the redesign of MediaSense affected the resource consumption and does it fulfil the requirements?"

7.2 Method Application

The research strategy used to evaluate the artefact was an experiment. The experiment was done by running an instance of the artefact and measure the memory usage while it is running. First the platform was started to see how much resources the platform use. When the platform was connected to the network, the researchers started to connect applications to the platform and take notes of how much memory every application was using. To test the old version of the middleware a node farm was used. A bash script was used to start several applications where every application had its own MediaSense platform.

Maximum four applications was connected to the platform when the researchers found a pattern in the resource usage. The data was then compared to data from the old artefact were an analysis was done to see if the new version use less memory resources. To see that all requirements were fulfilled different tests were done where every test had an expected result. All results was discussed between the researchers to address if result was as expected and to see if the requirements was met.

The experiment was done on an PC with operating system Ubuntu 12.04.2 LTS. The computer in use has a Intel Core i3-2350M processor and 8 GB Memory. To measure the resources the applications Gnome System Monitor 3.4.1 [13] and htop 1.0.1 [29] was used.

The non-functional requirements were not evaluated with a specific approach, they were discussed as the experiments took place to see that they were addressed and met. To test if the new artefact consumes less memory from the device one to four applications were run with both the old version of the middleware and the redesigned version. The results were noted and a comparison was then done. A test where platform and application were run on separate devices was also done to check the Gateway requirement.

7.3 Test Results

7.3.1 Functionality

Test	Start the platform
Produce	This was tested by adding the platform as a startup service on a linux computer. When the computer was started the platform started.
Expected Results	The expected result is that the platform starts and connects to the network without any applications connected to it.
Results	As Expected
Comments	This test shows that the requirement Platform as Daemon is met.

Test	Connect an application to the platform
Produce	This was done by starting a MediaSense application and connecting it to the background process from the previous test. The application registers itself with the platform and a connection to the platform is established. To see that the platform and the application is connected to each other the method getLocalhost() was called and the application printed this out in the output console.
Expected Results	The application connects to the platform and when the application register itself to the platform they are connected. When getLocalhost() is called the localhost for the platform should be printed in console.
Results	As Expected
Comments	This shows that the old functionality still exists and work in the new artefact.

Test	Connect several applications to the platform
Produce	Four applications were connected to the MediaSense daemon. All applications register themselves at the platform using the Interface method registerApplication.
Expected Results	All applications connects to the platform and connections are established without any errors.
Results	As Expected
Comments	This test shows that several applications can connect to the platform. This test also shows that the requirement *Several Applications* is met.

Test	Storing an UCI
Produce	One of the applications connected to the platform store an UCI by calling the method registerUCI.
Expected Results	A DuplicateUCICheckMessage is sent from the platform and a DuplicateUCICheckResponseMessage will be received when the UCI is stored in the network.
Results	As Expected
Comments	This test shows that the functionality still works. The methods which are called are from the provided Interface. This shows that requirement *Interface to applications* is met.

Test	Resolving an UCI
Produce	An application connected to the platform sends a ResolveMessage.
Expected Results	The application should call the method resolveUCI, causing the platform to send a ResolveMessage. When the message has been routed through the network a ResolveResponsMessage should be received by the platform and be passed to the application.
Results	As Expected
Comments	The methods that are called are the methods from the provided Interface. This shows that requirement *Interface to applications* is met.

Test	Sending a message
Produce	This was tested by building an application that sends NotifyMessages in response to a GetMessage. One of the connected applications sends a GetMessage and the application receiving this messages responds with a NotifyMessage.
Expected Results	A NotifyMessage should be received by the application that sent the GetMessage.
Results	As Expected
Comments	The methods that are called are the methods from the provided Interface. This shows that the requirement *Interface to applications* is met.

Test	Application crash with several application connected to platform
Produce	Connect several applications to a platform on one device. Make one of the applications crash by throwing an exception. A peer message was then sent to all applications connected to the platform.
Expected Results	Platform should still be working. No other applications connected to the platform should be affected by the crash. Messages should still be able to be sent and received by the platform.
Results	As Expected
Comments	This shows that the requirement *Application independent* is met.

Test	Send peer scope message
Produce	Sending a NotifyMessage from an application with the scope PEER.
Expected Results	All applications on the receiving node should get this NotifyMessage.
Results	As Expected
Comments	Requirement *Message with scope* is met.

Test	Send application scope message
Produce	Sending a NotifyMessage from an application with the scope APPLICATION and the application ID as an argument.
Expected Results	Application with the specific ID should get the notifyMessage.
Results	As Expected
Comments	Requirement *Message with scope* is met.

Test	Connect an application to an external MediaSense platform
Produce	The platform was started on one computer and an application running on another computer connects to the platform by getting its reference from the RMI registry.
Expected Results	Applications can communicate with the platform running on a PC.
Results	As Expected
Comments	Requirement gateway is met.

7.3.2 Resource Usage Measurement

Memory Usage Old MediaSense Version

Number Of Applications	Memory Usage	Total
0	0 MB	0 MB
1	70.4 MB	70.4 MB
2	68.8 MB + 70.4 MB	139.2 MB
3	68.8 MB + 71.4 MB + 73.2 MB	213.04 MB
4	68.8 MB + 71.4 MB + 73.2 MB + 73.1 MB	286.5 MB

Memory Usage New MediaSense Version

Number Of Applications	Memory Usage	Total
0	66 MB	66 MB
1	66.0 MB + 20.2 MB	86.2 MB
2	65. 4 MB + 19.5 MB + 19.5 MB	104.4 MB
3	65. 4 MB + 19.5 MB + 19.5 MB + 19.2 MB	123.6 MB
3	63. 2 MB + 19.5 MB + 19.5 MB + 19.2 MB + 20.1 MB	141.5 MB

CPU Time Old MediaSense Version

Number Of Applications	Memory Usage	Total
1	2.70	2.70
2	2.70 + 2.64	5.34
3	2.84 + 2.81 + 3.07	8.72
4	3.32 + 2.82 + 3.10 + 3.64	12.88

CPU Time New MediaSense Version

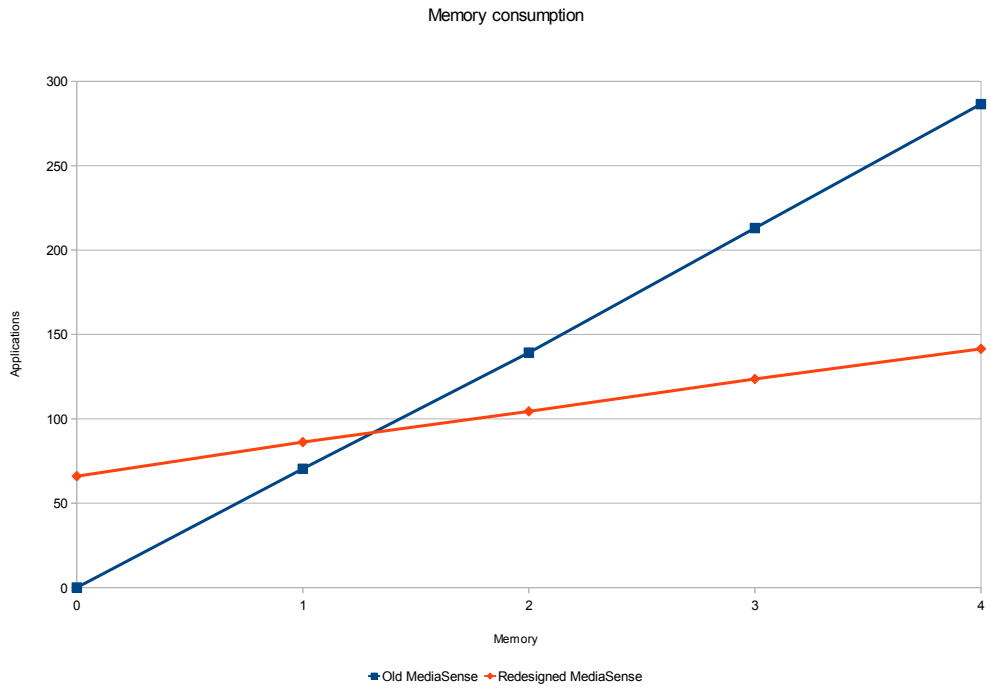
Number Of Applications	Memory Usage	Total
1	3.33 + 0.88	4.21
2	3.18 + 0.70 + 0.69	4.57
3	4.29 + 0.66 + 0.65 + 0.60	6.2
3	4.40 + 1.02 + 0.90 + 0.91 + 0.89	8.12

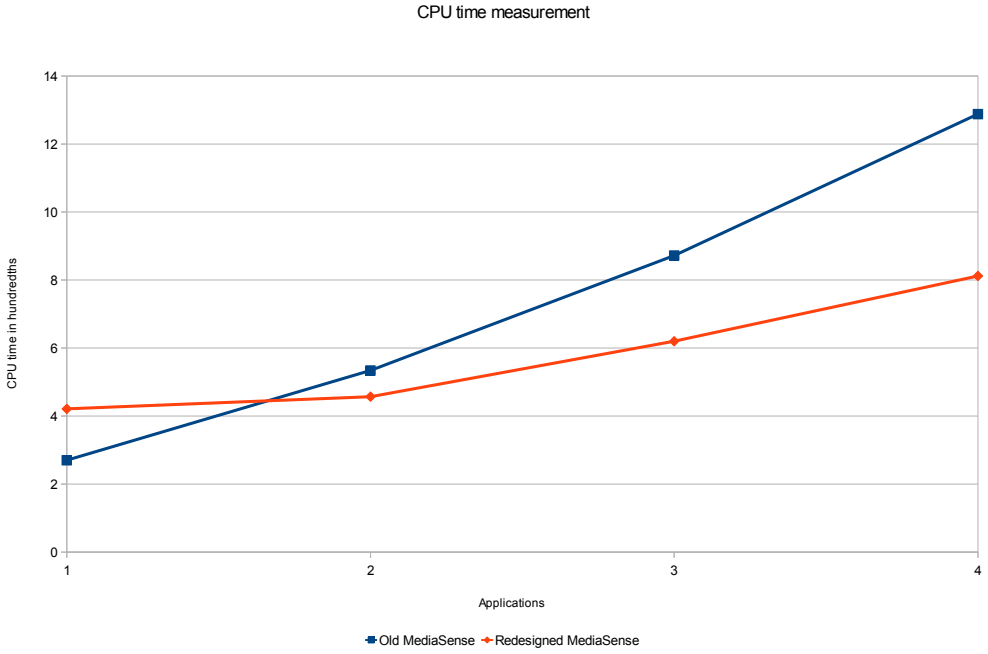
Threads Old MediaSense Version

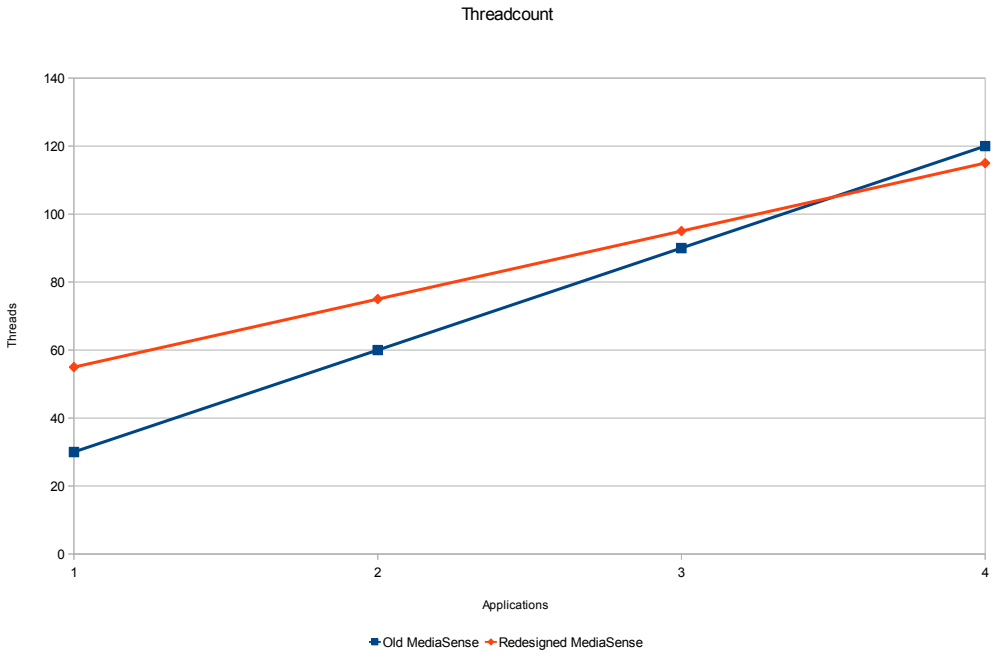
Number Of Applications	Memory Usage	Total
1	30	30
2	30 + 30	60
3	30 + 30 + 30	90
4	30 + 30 + 30 + 30	120

Threads New MediaSense Version

Number Of Applications	Memory Usage	Total
1	36 + 19	55
2	37 + 19 + 19	75
3	38 + 19 + 19 + 19	96
3	39 + 19 + 19 + 19 + 19	115







7.4 Analysis

After running the different scenarios the researchers believe that all the requirements was met. The non functional requirements was also met. The code was written with Java 1.5.

An object oriented style was followed the decision of using RMI instead of other RPC techniques made this requirements easy to follow. The overlay that was provided in the old artefact was not modified thus the lead developer preferred an unmodified network overlay and this requirement was met by the researchers. All functional requirements was tested with different scenarios as shown in method applications from this chapter. As shown from the memory measurement the platform still uses a lot of memory and if only one applications is running on a device there is no benefits of using the redesigned version of MediaSense. As shown from the test result the benefits of the redesigned version of MediaSense comes when more than one applications is running on a device. In the old version every instance of MediaSense with a small application with basic functionality takes around 60-70 mb memory. If the device is running n applications the minimum memory usage will be n*70. Notice that it is hard to measure how much memory an application is using in the old design because this version was applications invoked. When running the new version of MediaSense on a device with n applications, the background process which is the MediaSense platform with RMI support uses between 60 and 70 MB memory. Also in this case a minimal application with basic functionally is running on the device and it uses around 20 MB memory. This means that n applications memory usages will be 70+n*20 MB memory. With the new design of MediaSense the heavy layers of MediaSense is only necessary to be run once on every device. The conclusions is if more than one application is running on a device the new version of MediaSense will use less memory.

The CPU time is the amount of time a CPU is used for processing a computer program. This test was done by starting the platforms and connect applications to it and then run it in 5 minutes to see how much processing time the artefact uses. As shown in the table the new version of the artefact uses more processing time when the platform only have one application connected to it. Like the memory usage the benefits of the new artefact comes when more than one application is running on the device.

When running the new version of MediaSense the number of threads are more in comparison with the old version. The researchers believe this is because more threads is needed when RMI is used. As shown in the graph the new version uses less threads when four applications is connected to the platform. This means that even if the number of threads is more it can be effective to use when several applications is running on a device.

The new artefact also solves the problem that was explicated before where every running instance of MediaSense needs its own network port to communicate with other nodes in the network. With the common network layer only one port is needed to be open and all applications can use this port to communicate with other nodes. This makes the middleware more user friendly and no configuration is needed to run several applications on devices.

8. Discussion

8.1 Conclusion

This thesis has shown that the resource consumption of a middleware for Internet of Things can be reduced by running it as a daemon. The overhead caused by using RMI for the inter-process communication causes it to use slightly more memory and processor time when only running one application, this small overhead is vastly compensated for when running more than one application.

Running MediaSense as a daemon makes the resource costs for the platform and network overlay a one-time cost which makes it possible to run a lot more applications compared to the old version. As an example of a ubiquitous device the Raspberry Pi [14] was mentioned. The Raspberry Pi model B has 512 MB of memory which is shared with GPU. The default split of the memory is 64 MB for the GPU leaving 448 MB as RAM. The most popular operating system for the Raspberry Pi is a modified version of the Linux distribution Debian called Raspbian, which can be configured to use less than 10 MB memory. A moderate estimate of the operating systems memory requirements in normal use would be around 30 MB, allowing for a little overhead that leaves 400 MB of memory. The results showed that the old version of MediaSense required on average 70 MB of memory which would allow 5 MediaSense applications to be run on a Raspberry Pi.

The redesigned version of MediaSense required 60 MB memory for the daemon and then 20 MB per application. This allows for 17 applications running on the same Raspberry Pi. A reengineered distributed Internet of Things middleware running as a background process uses less resources when several applications are running on a device. This is one step closer to fulfil the Internet of Things concept where ubiquitous computers are pervasive in in the physical environment.

8.2 Significance And Originality

Usage of architectural middleware like RMI is common practice in all fields of computer science. The usage of RMI in MediaSense was done with consideration to resource consumption and shared resources to support ubiquitous devices which haven't been done before. In [42] published in 2001 it is concluded that RPC is harmful to ubiquitous computing. MediaSense uses the kind of asynchronous communication proposed in [42] between nodes but ubiquitous computers have evolved a lot in the last 12 years. In the case explored in this thesis it has shown that the RPC implementation RMI can be used with great success in ubiquitous computing.

8.3 Societal and Ethical Implications

This research has shown that a distributed approach to the Internet of Things middleware is viable on ubiquitous devices. As this makes immersive applications using context data feasible on mobile devices this could mean a big change in computer user behavior. Because MediaSense uses a distributed approach to store and share context data, the users data will be stored on computers other than their own. As it is today, MediaSense does not encrypt this data. Since the data is distributed this means there won't be a single vector of attack and specific users data cannot be as easily obtained.

8.4 Future Studies

In the future MediaSense could be ported to mobile operating systems such as Android and iOS. This could be done by using PhoneGap [40], a cross-platform framework for mobile apps with standards-based Web technologies like HTML, JavaScript, CSS.

To continue reducing the resource overhead in MediaSense, future studies can investigate how much resources the overlay uses and if it can be optimized to reduce the resource consumption.

Also future studies about the context information that is stored in the distributed network can be done. As mentioned data is not encrypted in MediaSense. To make the middleware more reliable an implementation with encrypted data in the network can be developed.

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] GNOME Foundation. The gnome system monitor. https://launchpad.net/gnome-system-monitor. Accessed: 2013-06-10.

[14] The Raspberry Pi Foundation. Raspberry pi. http://www.raspberrypi.org/. Accessed: 2013-06-10.

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

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

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

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

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

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

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

[26] MediaSense. Mediasense download. http://real3.miun.se/download.html. Accessed: 2013-06-07.

[27] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, DeQing Chen, T. J. Giuli, and Xiaohui Gu. Towards a distributed platform for resource-constrained devices. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 43–51, 2002.

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

[29] Hisham Muhammad. htop - an interactive process viewer for linux. http://htop.sourceforge.net/. Accessed: 2013-06-10.

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

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

[32] 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.

[33] 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.

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

[35] G. C. Roman. A taxonomy of current issues in requirements engineering. *Computer*, 18(4):14–23, April 1985.

[36] T Scott Saponas, Jonathan Lester, Carl Hartung, and Tadayoshi Kohno. Devices that tell on you: The nike+ ipod sport kit. *Dept. of Computer Science and Engineering, University of Washington, Tech. Rep*, 2006.

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

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

[39] 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.

[40] The PhoneGap Team. Phonegap. http://phonegap.com/. Accessed: 2013-06-10.

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

[42] David J. Greaves Umar Saif. Communication primitives for ubiquitous systems or rpc considered harmful.

[43] Jamie Walters. Ripples across the internet of things : Context metrics as vehicles forrelational self-organization, 2011.

[44] 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.

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

[46] 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.

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

Stockholm University

Department of Computer and Systems Sciences

Forum 100

SE-164 40 Kista

Phone: +46 8 16 20 00

www.dsv.su.se

Stockholm
University