DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Thesis type (Bachelor's Thesis in Informatics, Master's Thesis in Robotics, . . .)

Thesis title

Aly Saleh

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Thesis type (Bachelor's Thesis in Informatics, Master's Thesis in Robotics, . . .)

Thesis title

Titel der Abschlussarbeit

Author:	Aly Saleh
Supervisor:	Prof. Dr.-Ing. Jörg Ott
Advisor:	M.Sc. Teemu Kärkkäinen
Submission Date:	Submission date

I confirm that this thesis type (bachelor's thesis in informatics, master's thesis in robotics, . . .) is my own work and I have documented all sources and material used.

Munich, Submission date   Aly Saleh

Acknowledgments

Abstract

Contents

List of Figures

List of Tables

1 Introduction

1.1 IoT & Distributed Sensor Networks

1.1.1 Show how Iot is being currently used, its pros and cons

1.1.2 Give an idea about the devices used to make a distributed sensor network

1.2 Motivation

1.2.1 Show the need to explore Pervasive Computing

1.2.2 Illustrate why it might be better to distribute the data in some cases rather than accumulating it in a single server

1.2.3 Explain why Cloud Computing is not always the right solution in some cases

1.2.4 Explain the need to find IoT devices capabilities and limitations when used for data computation

2 Background & Related Work

2.1 Introduce Edge, Fog and Pervasive computing, how they are used in this context

2.2 Explain how sensor data data is modeled and distributed in the current published approaches

2.3 Illustrate what are the ideas and possible network mechanisms and protocols that could be used data transfer

2.3.1 Server To Server

2.3.2 Server To Device

2.3.3 Device To Device

2.4 Explain Opportunistic networks and SCAMPI architecture

2.5 Show other approaches in the literature

3 Approach

3.1 Requirements

3.2 Use Cases

3.3 Modeling of Input Sensor Data

3.3.1 Show how the different sensors have data been modeled to fit our requirements for further use in computations

3.4 Computation Model

3.4.1 Dealing with Dependencies

Before proceeding to examine the computation itself and explain how it is designed, we must first show what are the dependencies that the computation would need to execute. There are different types of dependencies; first are the software frameworks that the whole design relies on and must exist on each Node. These are the common libraries and systems that most of the computations would need to execute. That's why, these dependencies are shipped to each Node in our design, examples of these dependencies include the operating system, data store, node-red and any other standard or custom libraries that is needed to guarantee a successful execution. In addition to, SCAMPI which must be included to allow communication between Nodes. These dependencies need only to be shipped once while initializing the Node.

Second, are the dependencies that are specific to each computation such as additional scripts, data files or libraries. In this case, they cannot be shipped at Node initialization since we cannot know what are the custom dependencies any computation would need beforehand. Therefore, the design of the computation model allows a way to configure additional dependencies, which are sent accordingly to any Node that is going to execute this computation. This creates a bit of ambiguity because what if the dependency that is being shipped already is on the receiving Node, also what makes it more complicated, is that the Node does not know if it is an older version of the dependency or a newer one. Furthermore, what if there is a computation on the Node that uses an older version of the same library while the maintainer is sending a new computation with a newer version of the same library that is not backward compatible. However, there are multiple proposed solutions to remove the ambiguity and make the custom dependency shipping more concrete; one solution would be to give the dependencies different names according to their versions before shipping them, hence, any different version would not replace the existing ones. Another solution would be to design a system that links each running computation on the Node to its dependencies and once a collision appears, the new computation renames its dependency and uses the renamed one.

3.4.2 Dealing with Resources

Resources are a different type of dependencies which are also necessary for computations to run. However, they might differ or not exist at all on each Node, if one of the needed resources to carry out the computation is missing then it could be either dismissed or queued and that depends greatly on the type of resource. Moreover, the maintainers cannot make any assumptions about them, meaning, an assumption stating that each Node has a camera is not necessarily true. Since the resources cannot be standardized, each computation must specify the resources it is going to need, then a Node can check against its capabilities and decide whether it could carry out this computation or not. This kind of information is also known as computation meta-data. Resources may be classified into two main types explained below.

3.4.2.1 Hardware Resources

Hardware resources are attached to a Node such as cameras, temperature and gas sensors. Also, executing a computation on a Node missing this type of resource should have a lower possibility of being queued, since its highly unlikely that this hardware resource would be attached soon. The computation model suggests several ways to describe a Node resource capabilities; first by using a specification file that expresses the resources in a certain Node. Of course this approach has its drawbacks since if we attach a new hardware resource we must also edit the specification file correspondingly and that increases the manual work. Secondly, by using operating system commands to discover the attached resources each time a computation needs to be deployed on a system.

Moving on to consider the risk of computations acquiring the same hardware resource at the same time, for instance, two computations that want to take a photo at the same time. This is problematic because whichever computation acquires the lock on the camera first will succeed while the other will fail. Therefore as a resolution, the design proposes resource decoupling; instead of having the computation to ask a specific resource directly for information, the resource will always push its data to a database. Afterwards, the different computations could query the data from the database.

3.4.2.2 Computational Resources

The second type of resources is related to the Node performance, its power and memory capabilities. Computations vary in terms of resource consumption and hence a heavy computation should not be deployed to a Node which is already loaded. Considering that each computation model has meta-data describing its resource consumption, then it rather easy to decide if it is going to be deployed on a specific Node or not. Additionally, if it is not going to be deployed then it should be decided whether the computations is going to be queued or dismissed according to the possibility of acquiring the resource.

3.4.3 Input Output Modeling & Flow Composability

It is believed that development of smart pervasive computing devices that uses sensors and actuators are mainly grouped into smaller disconnected architectures and thus hard to create a composite framework in which all devices can communicate and integrate[GFT10]. In order to tackle this problem and to ensure that our computational model design is dynamic and flexible enough, we must ensure that our model is composable. By Composability we mean that computational models should be able to communicate, send and receive input and output data. Also, the models should be able to either communicate directly to each other via global referencing or through discovery in which they share the same interest. Moreover, input and output data structure should be modular to allow dynamic sending and receiving of data. Below we explain how the computational model design overcomes these challenges.

3.4.3.1 Controlled Vs Uncontrolled Communication

Controlled communication in our context basically means that we have one or more computation on more than one node, these nodes know each others global reference and wishes to exchange their input or output data. On the other hand, the uncontrolled communication suggests that there are several interested nodes in one or more computations and might not know each others global reference. To allow both types to communicate, our model proposes to use SCAMPI to transfer the message between them. In the case of controlled communication, then we add an attribute to the message with the receiving node global reference, hence, ensuring that only the node with the exact same global reference is allowed to pick up the message. In the other case of controlled communication, the global reference attribute is disregarded completely allowing all interested nodes to collect the message.

3.4.3.2 IO Specification

Turning now to consider the modularity of the input and output specification, as discussed above, we need the IO to have a specific structure that all nodes can understand regardless of the content. There are multiple ways in our design to specify how the IO data communicates which is explained below. But, the key idea however is that every type of IO communication can be described as a form of node-red flow in one way or another.

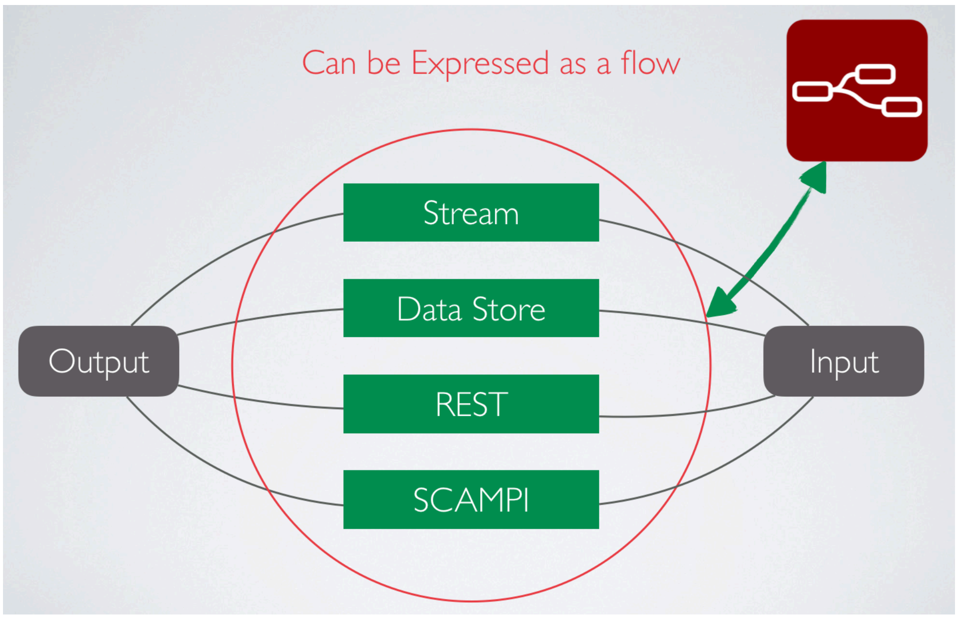

Figure 3.1: All IO types could be expressed as a flow

- The first way allows data communication between computations of the same node through a database. One computation writes interesting data into a specific table with locally unique name in a database. Then, any other local computation which needs to use this is data is allowed to fetch it from this table. Since node-red allows database configuration from inside the computation flow there is no extra effort in writing an additional script to write data into a database. The example in 3.2 shows a flow that takes an image and then store it in the database, while the other pulls the data from the database upon receiving a request on a specific URL.

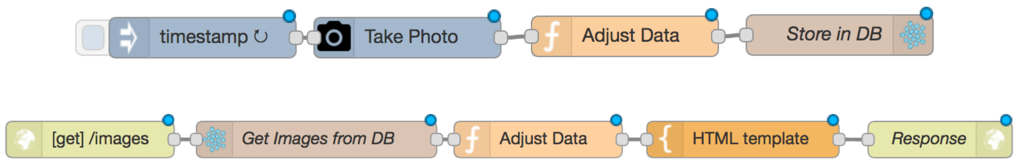

Figure 3.2: Two separate computational flows describing the IO through a database

- Another way is to use SCAMPI publish-subscribe messaging pattern to communicate, this way computations does not have to be on the same node, in fact, they do not have to be connected at all. The reason for that, is because SCAMPI can connect to mobile devices passing by, hence allowing them to make the data transfer to another nodes. The node which generates the data publishes its resulting data to a generally unique topic, therefore any node interested in the data could simply subscribe to that topic and process the data accordingly. The figure 3.3 shows two flows as an example of this method, the first flow generates a timestamp and publishes it to SCAMPI. Then, on any node, if we have the second flow available SCAMPI could get the data and then any computation could be done with the data on node-red.

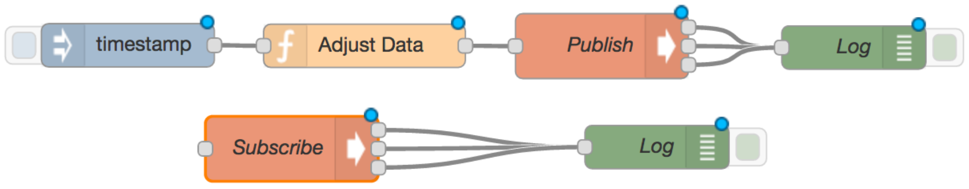

Figure 3.3

- I/O spec design for databases for two composable flows - Composability has the orthogonal issue of matching input/output "Controlled" IOSPec

3.5 Moving Data

3.5.1 Data Streaming

we can ask the hight performance units to stream from cameras no low devices. but how to do it ?

3.5.2 Explain data distribution among several nodes to apply pervasive computing

3.5.2.1 Input Meta-data

is it local, provided or collected data.

3.6 Networking

3.6.1 SCAMPI

3.7 System Design

A System Design can be broadly described as an architecture of the system, which includes an explanation of each and every hardware component of the system, the connection between these components if there is any, and the data flowing between these components. Moreover, it provides a wide glimpse of the whole system but not its exact functionality, hence, giving a simple understanding of the architecture without jumping into much detail.
Initially, the components of the System Design is introduced, then, the connection between these components is shown, and eventually, the flow of the data is pointed out.

3.7.1 Components

Below, each component of the proposed system design is explained.

3.7.1.1 Node

A Node is one of the core components of this design, it is a small computer device of low storage and computation capacity compared to nowadays portable computers, commonly a *Raspberry Pi* but could be any other device. It is connected to several sensors which typically detect certain changes in the environment and converts it into digital data, for instance, Gas sensor, Temperature sensor or a Camera. Then, the device either stores the data into a local database, performs a computation locally, does both or even asks other nodes to do computation instead, however, an assumption about which sensors or specifications does a specific node possess can not be made, meaning, each node might not have the exact number or types of sensors because each node may be deployed in a different timing or context. Thus, each node has a configuration file specifying its capabilities. A typical node is shown in figure 3.4

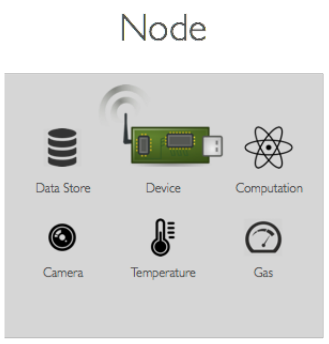

Figure 3.4: A typical node in the system

3.7.1.2 High Performance Units

A high performance unit is a device with massively parallel architecture designed to handle multiple tasks at the same time, thus observed to be much faster and more efficient than a Central Processing Unit *CPU*, and in turn, has higher computation capabilities than the CPUs in the proposed system nodes in 3.7.1.1. An example of a high processing unit is a Graphics Processing unit *GPU*.

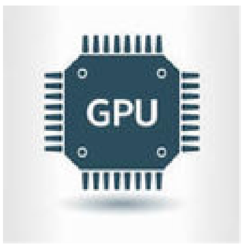

Figure 3.5: Figure denoting a Graphics Processing Unit GPU

3.7.1.3 Network

A Network in this design is a set of connected components which are capable of communicating and therefore allowing data sharing between them.

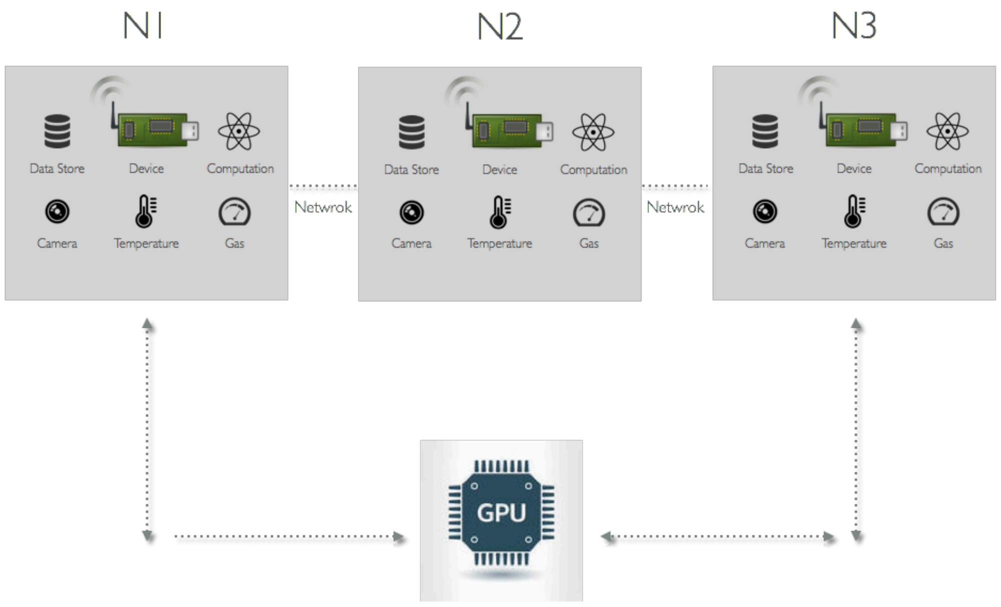

Figure 3.6: A network consisting of three connected nodes and a GPU

– TODO: Emphasis the difference between persistent and non persistent network links in system design.

3.7.1.4 Mobile Device

A Mobile Device in this context is any device that can connect to the network containing the nodes and is allowed to carry data from one network to another, hence, allowing a form of data sharing between networks or nodes which are not connected.

Figure 3.7: Figure denoting a Mobile Device

3.7.2 Connectivity and Data Flow

A Network described in 3.7.1.3, is a simple form of connectivity between components, however, components and specifically nodes are not necessarily connected, sometimes they are just a standalone component that cannot share any information via direct connectivity, also, networks could be disconnected as well, meaning, a network might not be connected to the whole system, thus, is a standalone network. In these cases, a mobile device could help in carrying information and data between these disconnected nodes or networks.

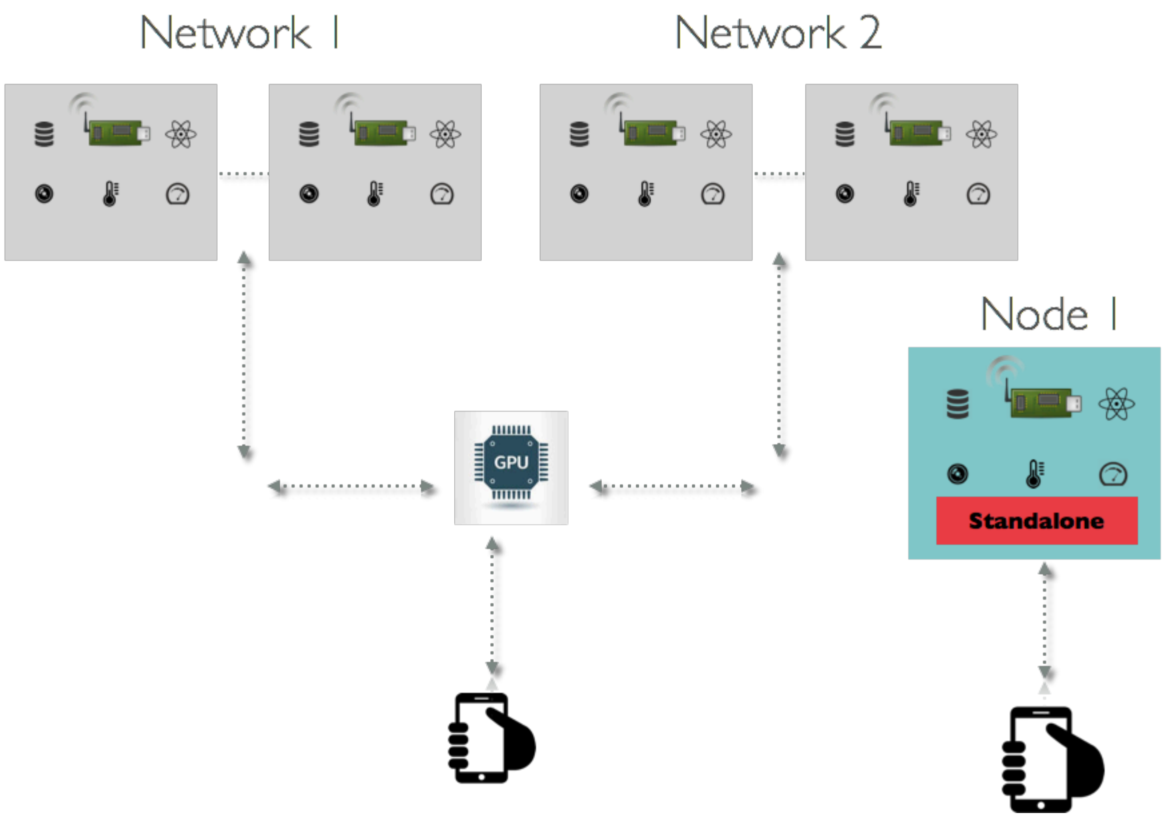

Figure 3.8: Two networks connected with a GPU and one standalone network

3.8 Summary

4 Evaluation

4.1 Implementation

4.2 Use Cases

Design of two, three or more use cases, however, implement one or two.

4.3 Performance Tests

4.4 Limitations

5 Conclusion

5.1 Summary

5.2 Future Work

5.2.1 Streaming API

Bibliography

[GFT10] D. Guinard, M. Fischer, and V. Trifa. “Sharing using social networks in a composable Web of Things.” In: *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. Mar. 2010, pp. 702–707. DOI: 10.1109/PERCOMW.2010.5470524.