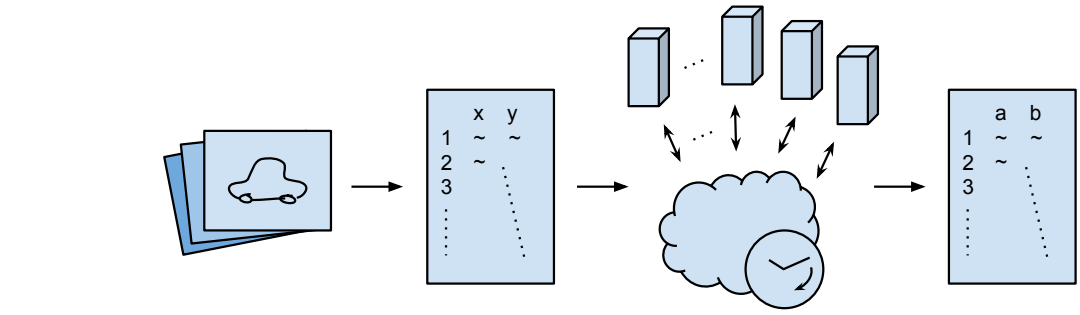

Implementing a cloud-based scalable dynamic model simulator

Master of Science Thesis in Computer Science - algorithms, languages and logic

DAN JOHANSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, January 2015

Implementing a cloud-based scalable dynamic model simulator
DAN JOHANSSON

Examiner: Joachim von Hacht

ISSN 1652-8557
Department of Computer Science and Engineering
University of Gothenburg
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, January 2015

Implementing a cloud-based scalable dynamic model simulator
Master of Science Thesis in Computer Science - algorithms, languages and logic
DAN JOHANSSON
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

This thesis describes the integration of a cloud service into an existing desktop application to increase performance and add new functionality made possible by distributed computing.

The prototype successfully integrates a cloud service's virtual machines into the application using a RESTful approach that create an API for clients to use.

The peer-to-peer architecture is discussed as an alternative to a RESTful approach but is ultimately turned down in favour of a RESTful API to support thinner clients.

A client makes a request to the API which in turn communicates with the cloud service provider which supply with computing power. The request is processed and the result is displayed in the client application. The approach is deemed successful and a solid way to create a service using distributed services.

Keywords: Cloud services, cloud providers, RESTful APIs, dynamic models, Modelica, simulation strategies

Acknowledgements

I would like to thank Modelon AB for the opportunity to perform this master's thesis.

I would especially like to thank Iakov Nakhimovski at the Modelon office in Gothenburg for providing supervision and feedback.

Finally, I would like to thank my examiner Joachim von Hacht for his help during this thesis.

Contents

1 Introduction

1.1 Background

Computer simulations is a way to reproduce the behaviour of another system that has been modeled to represent a scenario. It is a way to explore and get better insight into how a complex system performs when put under certain constraints or other conditions. The runtime of a single simulation varies greatly from a couple of seconds up to several days. The time it takes is greatly dependent on the complexity of the systems involved but also on the capacity of the machine that actually performs the simulation.

Computationally heavy work can be improved by having more computers to help out. Often this is done by letting them work together on a particular simulation. It can be cumbersome and expensive to manage a set of physical computers dedicated to simulation. However, it is possible to rent a set of computers and letting them do the work.

Cloud services have emerged as a business to manage and deliver services across the Internet. Services where it is possible to run complex simulations but also a range of other applications that require some hardware configuration. However, cloud computing specifically revolves around computational work and is as such suitable for simulation. With cloud computing it is possible to rent virtual computers and prepare them with data for the simulations. The price of renting virtual computers is often less than of owning an equivalent physical setup. The low cost makes cloud computing approachable to others who may not have the hardware dedicated to perform computationally heavy work.

The pricing of cloud services comes with a range of options depending on specific needs of the user. For instance, consider Dropbox which is a cloud storage service. They offer different plans depending on what needs the user has. Their cheapest plan is free of charge and gives the user 2GB of storage. The next tier offers storage space starting at 100GB for a fee.

Modelon is a company that provides solutions in physical modeling, simulation, and optimization and often work with dynamic models of complex systems. Models are created using Modelica which is an object-oriented modeling language. Modelon is now looking at cloud computing as a means to increase productivity on both their own and their customer's behalf.

1.2 Problem description

Modelon have created a plugin to Microsoft Excel that allows a user to work within a familiar environment. The plugin wraps an application that performs the actual simulation. The application is only able to run simulations on Microsoft Windows operating systems due to dependency restrictions. Being platform dependent leaves out a great deal of users and potential customers.

The application consumes the majority of the computer's resources when it is simulating which takes a notable toll on the performance of the local machine. This is something that is tolerable for shorter durations but may impact other work over time.

Heavy resource consumption can be difficult to do anything about since it depends on a lot of things but a reason why it would be desirable to either decrease the amount of resources used or shift the way the resources are used is that the heavy usage puts a limit on the computer's capabilities. Meaning that it will impact other tasks currently at work such as working within an IDE, preparing new models, and so on.

Additionally, the running time of a simulation is related to the hardware setup

which raises a question regarding the cost of said setup. In order to solve complex problems faster you prefer some computer components over others, such as a fast CPU. Such components tends to have a matching price tag which makes it reasonable to ask how to balance the processing power.

1.3 Purpose

This master's thesis aims to resolve the issues mentioned in the problem description regarding the existing Modelon plugin by creating a prototype that makes use of cloud services. The prototype will add functionality to the plugin and will keep the existing workflow of the user intact.

1.4 Scope

The result is limited to be a functional prototype in the sense that it should be able to maintain the existing workflow and produce the same results when it comes to simulation.

An important aspect of cloud services is security. Security regarding cloud services and their interactions is not a part of this thesis. There will be no mentions regarding practices and concerns for different computing concepts when it comes to security. When it comes to the implementation of the prototype it will assume that industry solutions for encrypted communications and similar is sufficient.

In the same spirit any other software principles such as reliability, availability and similar is left out.

1.5 Method

The way this work is conducted is through literature studies where a lot of the information regarding a cloud service is available from its provider. Some cloud service providers have white papers that give greater insight into their specific service.

In addition to modifying the plugin, an API will be developed that allows the prototype to work with the cloud service provider. An application wrapper will also be developed in order to facilitate simulation with the API.

1.6 Related work

One approach to computationally heavy tasks is to utilize the GPU instead of the CPU. Furthermore a GPU is optimized for a certain type of calculations which may be faster compared to a CPU. [Kre09]

2 Theory

2.1 Distributed computing

Distributed computing [KS11] refers to the use of distributed systems to solve computational problems. A distributed system is a collection of independent systems that work together to solve a problem. A distributed system can be identified from the following features:

- No common physical clock
- No shared memory
- Geographical separation
- Autonomy and heterogeneity

Historically, distributed systems appeared relatively recently considering how long computers have been around. As time progressed they become both smaller and cheaper. That led to people being able to afford and put several computers in the same room. In addition, they also increased in processing power meaning that the time spent on network operations decreased and the resources could be better used solving a problem. Combine that with an increase in bandwidth across computer networks and it became feasible to connect computers together for computational purposes. [Krz03]

There are several reasons for applying distributed computing concepts to a problem. The most apparent may be that an application requires distributed computing. Meaning that it can't function properly otherwise. An example of such an application is where a machine requires data only available to another machine.

Another reason is that it is more practical. It may be that a single machine can solve a problem perfectly fine but it is beneficial to use a distributed system. For example it may be more cost-efficient to use a set of lower end computers as a cluster in order to achieve a certain level of computing power instead of using a single computer. A distributed system can be more reliable than a non-distributed system due to there not being a single point of failure within the distributed system. Additionally it may be easier to scale and manage a distributed system rather than single machine system.

This section describes the different principles of distributed computing with additional focus on concepts that are used by companies as a means to provide computer resources.

2.1.1 Client-server model

Networks where certain computers have dedicated tasks, such as providing services to other computers in the network, are called client-server networks [GB12]. Often clients and server communicate over a network but both client and server may reside on the same system. A server runs one or more applications which share their resources with a client. A client makes requests to the server's services and does not share any resources. Clients initiate communication with servers.

An advantage of this architecture is how the client and server applications are encapsulated making maintenance easier. In addition to maintenance, upgrades to server applications become simpler when the data is centralized. However, if a server were to go down the clients would be at a standstill until the server is up and running again. Such failures can be mitigated with load balancers and failovers to ensure that the client experience is consistent while the issue is dealt with.

A typical usage for the client-server architecture is websites or web services where the client sends a request to a service and the service responds with for instance a file or a picture (file server).

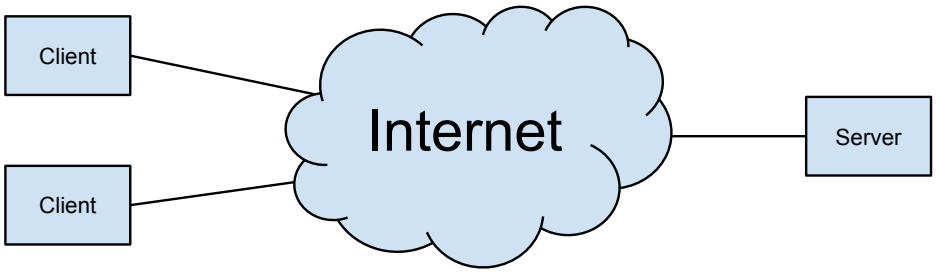

Figure 2.1: *An example of a common client-server relationship.*

2.1.2 Peer-to-peer model

A peer-to-peer network [GB12] is a decentralized network where individual nodes (also known as peers) acts as both the client and the server. In peer-to-peer networks, tasks are shared amongst the peers who each make a portion of their resources available to other network participants. Such resources could be processing power or disk storage. All without centralized coordination from a server.

Unlike the client-server model described in 2.1.1 the peer-to-peer network is unaffected if a node were to shut down due to the decentralized characteristic of the network. Nodes in a network are aware of the other nodes such that the behaviour is adjusted whenever new nodes connect or disconnect. This makes a peer-to-peer network very stable.

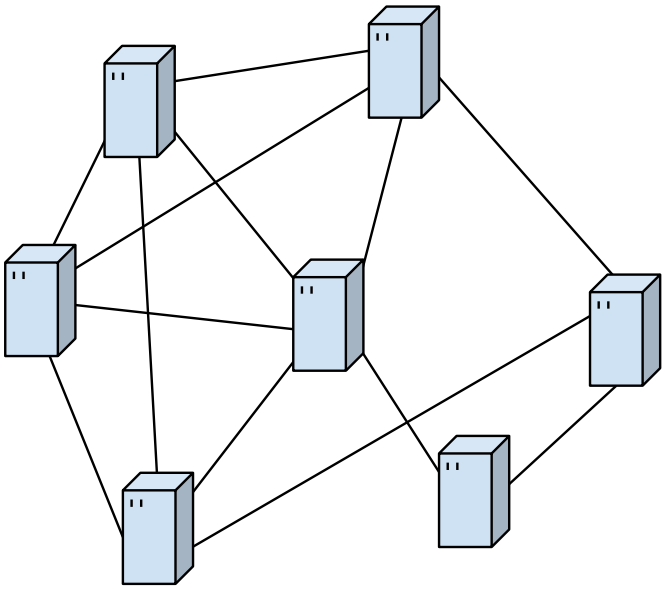

Figure 2.2: *An example of a swarm with several nodes. The lines connecting the nodes correspond to some form of communication.*

Making changes to the network can be time-consuming since the changes in the software have to propagate to each node unlike the client-server model whereas an update in a centralized approach is effective at once.

Napster [JP99] is a typical example of a peer-to-peer application in which users share files with each other, often music files.

2.1.3 Cluster computing

Cluster computing [GB12] became a reality when users no longer could fit all of their work on a single computer. The date cluster computing was introduced is unknown but according to Greg Pfister's book In search for clusters it wouldn't be surprising if clusters didn't exist in the 1960s or late 1950s.

A computer cluster is a set of interconnected computers that closely cooperate to provide high performance computing capability. To be more specific it consists of a large number of regular computers that each run their own instance of an operating system with additional software that enables high performance distributed computing. The hardware used in the computers that form the cluster often consists of commercial off-the-shelf components that yield reliable and in many cases cheap setups but there are no rules as to what hardware may be used in a setup to be called a cluster.

The simplest cluster available would be two computers connected to each other where both computers work closely together on a single problem so that the two computers can be viewed as a single system.

The growth of computer clusters has been closely tied to the growth of computer networks since one of the primary factors of developing networks was to connect computational resources.

Computer clusters are used to perform tasks that require computing capability that are not found in easily accessible computers such as regular desktops, or even high-end desktops for that matter. Tasks that require such computing capability involve, but is not limited to, rendering computer graphics, compiling code, and general computational intensive applications.

The biggest benefit of cluster computing is the performance to cost ratio, meaning that it is possible to get something that is equivalent to a single computer with high computing capability using several computers with less computing capability.

Also a benefit of the structure of the connected computers is the ease of augmenting certain parts of the cluster. Be it the number of computers connected, memory capacity per computer or number of processors per node.

The downsides are connecting many computers requires a lot of physical space and the resources to keep them all up and running can be extensive. The maintenance of the cluster may at times be equivalent to maintaining the same number of independent machines.

One famous example of a cluster is the Jeopardy champion more known as Watson [UMB11]. The IBM supercomputer is a cluster consisting of ninety IBM Power 750 servers that together can process 500 gigabytes of data per second.

2.1.4 Grid computing

Grid computing [Fos02] refers to a more loosely coupled structure than cluster computing and is geographically dispersed. Applications of grid computing utilizes spare personal computer resources. The name grid originated as a metaphor for getting access to computational power in the same way as the electrical grids do.

There are a few notable public distributed computing projects such as Folding@home [Pan00] which connects regular personal computers through the Internet. Each computer has the Folding@home software installed that lets a computer participate. A cycle begins by connecting to the Folding@home servers and fetching a workload. This workload is a fraction of the simulation that needs to be done and is what that specific computer is to simulate. The application only performs the simulation whenever the client is idle so that not to disturb the user when using the computer. When the work is complete the client sends back the result and receives a score related to the size and complexity of the workload and starts a new cycle.

This particular project forms a supercomputer by connecting computers all over the world to lend their spare resources. All that is required for a client to participate

is their specific software and a connection to the Internet that is used to download the unit of work and to send back the results.

2.1.5 Cloud computing

Cloud computing [KS11] is distributed computing made accessible. A cloud service provider dictates the term of the computer service which limits a consumers control over some parts. Such things could be that several customers share resources such as RAM or processing power. Most times it is possible to pay more and in turn receive more capacity or more control of the resources available.

Below are four sections corresponding to the different levels of services that are considered to be a part of cloud computing. It is important to note that these descriptions characterize the biggest factors of each level. It is up to each cloud service provider to decide how they want to implement or provide services at a given level. This makes it difficult to clearly tell the levels apart.

Cloud computing is inherently more environmentally friendly due to the fact that it serves more than a single user. This removes the need for a user to purchase, set up, and power a data center of its own. Leaving the user to pick and choose from cloud providers that conform to their own ideals when it comes to renewable energy, environmental health, and work ethics.

A good example are Rackspace [Rac14] which details how the company reflects upon renewable energy and how they work alongside environmental organizations to further cement their focus on sustainability. Google [Goo14b] is a company that complies with certain sustainability-related standards regarding all their services such as ISO50001.

Infrastructure as a service

Infrastructure as a service [LV12], or IaaS, is the most basic of the four models since what you get is a virtual computer; an empty computer (in most cases) that needs an operating system and software and properly configured runtime environments before any applications can run.

An important note is that while this model is more flexible than the others it also requires more attention since the virtual machine is completely the consumer's responsibility. Therefore, in order to properly maintain it, there should ideally, be a system administrator.

The price for services of this type is often very flexible. In most cases there is a possibility to pay per hour. That is, each feature has a tied-in cost to it and if a user consumes this feature for a number of hours then pays accordingly.

Managing an instance on this level is either done through an administrative web interface or through the cloud providers API. Configuring the instance is dependent on what kind of operating system that is installed. For Windows platforms a virtual desktop is common and for Unix systems an SSH connection can be used.

Platform as a service

In the platform as a service [LV12] model the cloud service providers manage a computing platforms where operating systems, programming environments, databases, and web servers are already in place. They essentially provide the developers with all they need to deploy their applications. This differs from the previous infrastructure as a service model in the sense that a developer on this level can focus on the application itself instead of the environment since the cloud service provider take care of any updates to the system.

One of the most appealing features of the platform as a service model and cloud computing in general is the possibility to scale applications. For instance, if users of a web application would increase, it would be possible to scale the application to

meet the demands of every new user without there being any noticeable change in the user experience.

The pricing in this case is based upon the amount of resources and the time that is used. It is often possible to seamlessly manage the resources such as increasing the storage space.

Software as a service

With the software as a service [LV12] model the cloud service providers have already installed the software that is to be used in the cloud. In addition to setting up the environment they also operate and maintain it. This means that whenever a security flaw is discovered it is up to the cloud service provider to patch it.

The cloud user usually accesses the application through a client and the user doesn't have to be concerned with the infrastructure or the application software. The client can appear in many shapes but the most common clients these days are web browsers. An example is Google Apps which is a collection of software for businesses such as the popular mail client Gmail, a document writer, and spreadsheets. Each accessible from a web browser.

Applications of this type are often designed with some kind of variation on the multi-tenant data architecture. The defining feature for this specific architecture is that multiple users use the same application instance (hence the name). For an application to properly embrace this principle it requires to partition (virtually) data configuration and provide a customized experience to a user or an organization.

An example of such an application is Google Docs where users can sign up and create documents and spreadsheets online. Multiple users can edit their own documents privately and customize their own experience of the online application.

The pricing for this kind of service is often proportional to the number of users connected to the application or related to some quantifiable factor such as storage space or bandwidth consumed. Some services provide value-adding features such as administrative tools.

Network as a service

With the network as a service [PW12] model users have access to additional computing resources in the form of bandwidth, switches, and routers. Cloud users can utilize these features for custom forwarding, load balancing, or packet inspection.

One of the problems that this service solves is how to effectively join workplaces that are scattered across the globe. There is in many cases no need for outside help if all employees are at one location but the complexity of networks increase as more physical locations are involved and the goal is to maintain that initial level of simplicity.

The pricing here is similar to the other models where a user pays for what kind or quantity of resources that is consumed but it is more common that you pay a fee for each person that will use the service.

2.2 Virtual machine

A virtual machine [SN05] is a software implementation of a computer (machine) that behaves like a physical counterpart. A virtual machine can be implemented as a system virtual machine or a process virtual machine. The former supports a complete operating system and emulates existing architecture. These are used for either providing a platform to run programs on where the real hardware is not available or for having multiple instances of virtual machines (known as hardware virtualization). A process virtual machine is designed to run a single process and is closely tied to a programming language. Such virtual machines are built with the purpose of providing program portability and flexibility.

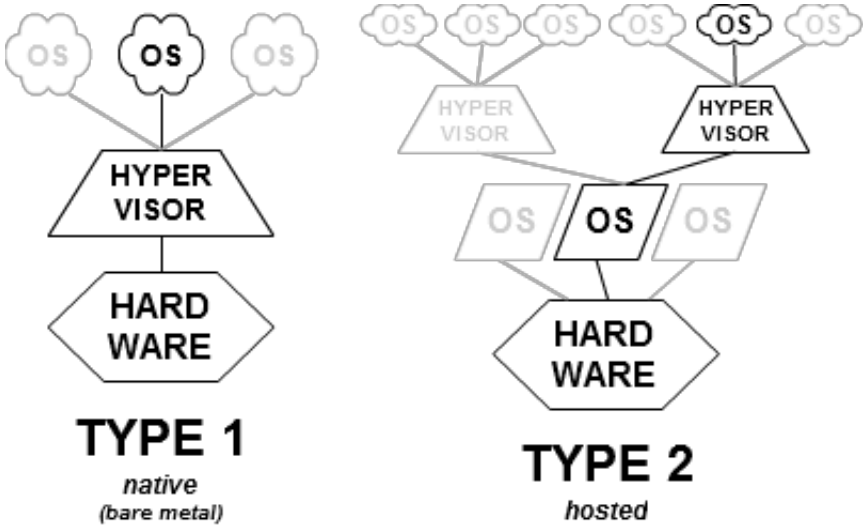

Figure 2.3: *Abstraction of a hypervisor*

Essential characteristics of virtual machines are that applications or processes running inside then cannot break out of the virtual environment.

The virtual machine is managed by a hypervisor. A hypervisor is a piece of software that runs on a physical machine that can operate and manage multiple virtual machines or guests as they often are called in this context. That means that several people can share the same hardware but be completely unaware of each other. The maintainer of the physical machine have total control over the hardware and the hypervisor but none regarding a consumer's virtual machine and vice versa.

In order to deploy an application on a virtual machine there has to be an operating system installed. When created as part of a cloud service the provider supplies the consumer with the operating systems. From there on there are primarily two ways to interact with it. The first being an administrative interface created by the service provider in order to do tasks such as terminate, reboot, and any other means that affect the properties or state of the actual virtual machine rather than the operating system and any software installed. The second way to interact is with a connection to the virtual machine such as SSH or a remote desktop connection. With a connection such as this it behaves as a regular desktop and it is possible to install applications and set up environments for applications

2.3 Web service

A web service is a method of communication between two network connected devices. The communication is often realised using HTTP requests with a service related message consisting of data represented by, for instance, XML or JSON.

The goal of the communication is for one machine to provide a service to the other machine, such as streaming media content.

2.4 Plugin

A plugin is a piece of software that adds a specific feature to an already existing application. Applications that support the use of plugins may allow for third-party developers to create functionality not present in the initial application and package it as a plugin. [Wik14]

2.5 API

An application programming interface, API, specifies the operations, input and output of a software component. The main purpose of an API is to define a set of functionalities independent of respective implementations.

An API specification can appear in many forms. One of the more common forms are as a library to a programming language where developers can access the functionality provided by the library.

For instance, web APIs are often defined as a set of HTTP requests along with a definition of the structure of the response messages. These response messages are usually formatted in XML or JSON. Web APIs can often be found working with a web service.

2.6 REST

Representational state transfer [Fie00], or REST, is an architectural style for distributed hypermedia systems. An example of a distributed hypermedia system is the World Wide Web, which is the largest implementation of a system conforming to the REST architecture style.

The architecture style provides a set of constraints that, when applied as a whole emphasizes scalable component interactions and interface generality. But also independent deployment of components, and intermediary components to reduce latency, enforce security, and encapsulate legacy systems. These architectural properties of REST are realized by applying a set of constraints to components, connectors, and data elements.

The architectural constraints are:

Client-server
: This model is explained in greater detail in section 2.1.1.

Stateless
: Each request from the client to the server contains all the information necessary to carry out the request. The session state is held by the client. There is no client context stored on the server.

Cacheable
: Responses must be labeled as cacheable to prevent clients from reusing state or erroneous data in future requests.

Layered System
: A client cannot tell whether it is connected directly to the end server or any intermediary servers.

Uniform interface
: The central feature of REST. The uniform interface simplifies and decouples the architecture allowing each part to evolve independently.

Code-on-demand (optional)
: Servers can extend the functionality of a client by transferring executable code. An example is client-side scripts.

The REST architectural style can also be applied to the development of web services. A web service is called RESTful if it conforms to the architectural constraints outlined above.

2.7 XML

XML stands for Extensible Markup Language and is a language with a set of rules defined to create documents that are readable by both humans and machines. XML is widely used when creating representations of arbitrary data structures for instance in web services. [W3C08]

2.8 JSON

JSON is an abbreviation of JavaScript Object Notation and is an open standard for transmitting data using human-readable text in key-value pairs. As an alternative to XML, the primary use of JSON is to transmit data between a server and an application.

When comparing JSON with XML in terms of size of messages JSON often wins with more lightweight messages. However, using a compression algorithm there is little difference between JSON and XML in terms of message size. [Int13]

2.9 FMI and FMU

Functional Mock-up Interface, FMI, defines an interface to be used in simulations of multi-domain systems [Pro14].

To create the FMI standard, a large number of software companies and research centers participated in a project called MODELISAR. The goal of MODELISAR was to improve the design of systems and of embedded software in vehicles. The outcome was the standard FMI. Since the end of MODELISAR the FMI standard is maintained by the Modelica Assocation.

Models created within a modeling and simulation environment that implement FMI can be exported as a software model called FMU, Functional Mock-up Unit. FMUs are distributed as zip files but with ".fmu" as a file extension. Each FMU contains several files.

- An XML file that contains the definition of all exposed variables and other static information.
- Model equations are provided with a small set of C functions. These C functions can be provided in source or binary form.
- Additional data such as bitmaps, documentation files, maps and tables needed by the FMU, and any dynamic link libraries that are utilized.

An FMU, as seen in figure 2.4, can expose input and output parameters that are available during simulation.

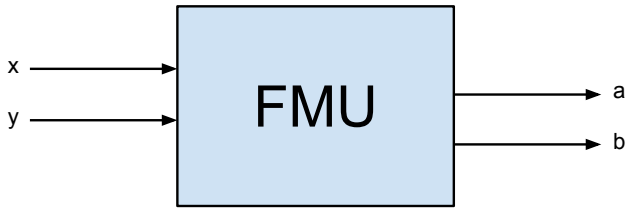

Figure 2.4: *An abstraction of exposed parameters of an FMU model.*

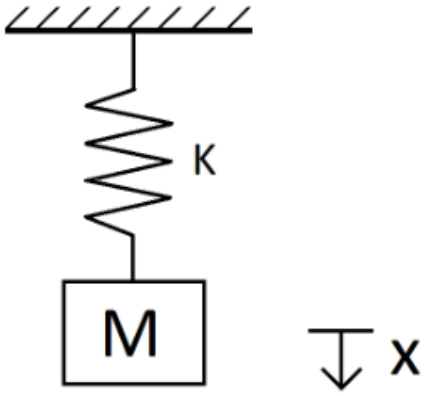

Figure 2.5: *Mass-spring system.*

2.10 Modelica

Modelica [Ass14] is a language used to model complex, multi-domain systems in an object-oriented and component based way. Classes are primarily built with equations. Unlike other programming languages where equations typically describes assignment. In Modelica equations describe physical properties. The equations are acausal meaning that the user does not have to define the order of the variables as long as the system is properly defined. The code below shows a mass-spring system modeled with Modelica.

```
model massSpring

    import SI = Modelica.SIunits;

    constant Real k = 2;
    SI.Acceleration a;
    SI.Velocity v(start=0);
    SI.Distance x(start=0);
    constant SI.Mass m = 1;
    constant SI.Acceleration g = 9.81;

    equation
        a = der(v);
        v = der(x);
        a*m = m*g-k*x;

end massSpring;
```

There is a free Modelica Standard Library [OE01] that includes a large number of components for a wide range of areas such as mechanics or electrical circuits. In addition to these it is possible to create custom components and libraries.

Working with Modelica is often done within an IDE called Dymola [AB14] (figure 2.6) which is developed by Dassault Systèmes. Dymola provides a graphical interface to intuitively create and display models in manageable way. Once a model has been created and is ready for simulation Dymola prepares a simulation environment and displays the results afterwards.

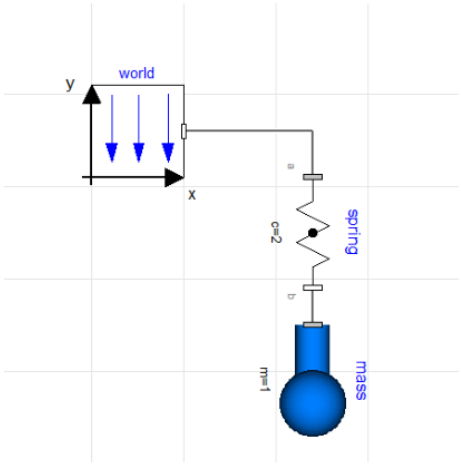

Figure 2.6: *Visual representation as seen from within the IDE Dymola*

2.11 FMI Add-in for Excel

The FMI Add-in for Excel, as seen in figure 2.7 (b), is a plugin to Microsoft Excel that enables the user to load an FMU into an Excel spreadsheet, figure 2.7 (a). From Excel the user can create a set number of test cases, called an experiment, and specify what the parameters should be available in the experiment. The test cases are simulated one at a time on the local machine and the result of the experiment (each test case) is shown in the spreadsheet.

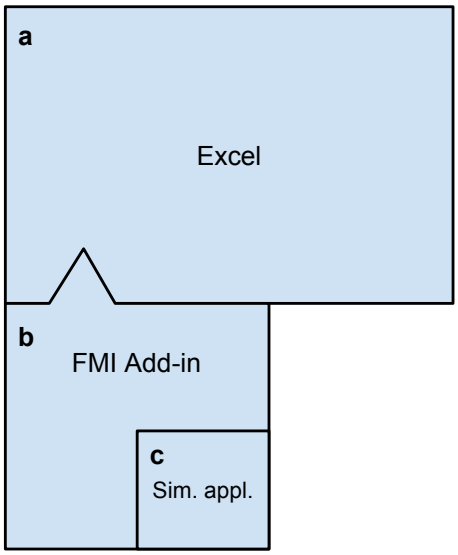

Figure 2.7: *Abstraction of how the existing components are arranged.*

The plugin itself contains a simulation application, figure 2.7 (c), that is used to perform the simulations. The simulation application, figure 2.8, takes an FMU and a test case as input and produces a result file.

Figure 2.9 shows the workflow of a user working with the plugin. The user begin by selecting an FMU. The FMU is then loaded into the plugin where the user specifies the desired parameters and creates a number of test cases. Once done the plugin lets the simulation application process each test case. When all test cases

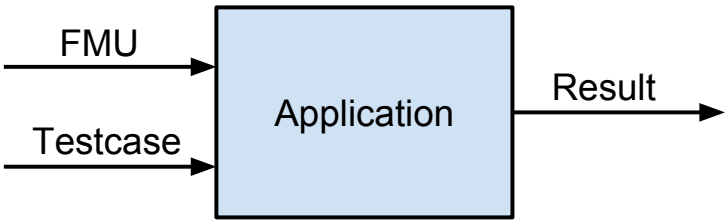

Figure 2.8: *Simulation application input and output.*

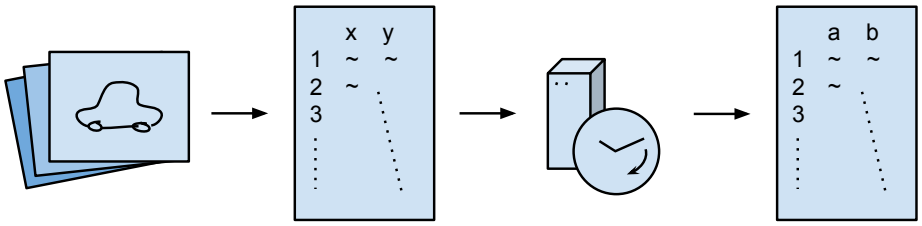

Figure 2.9: *FMI Add-in for Excel workflow.*

are finished, successfully or otherwise, the result for each test case is displayed in the same spreadsheet.

Figure 2.10 is what the spreadsheet looks like after an FMU has been loaded into the plugin.

As shown in figure 2.11 each column represents a test case whereas the entire spreadsheet is called an experiment. For each column (test case) a file is created with the information regarding that specific test case, for instance input values.

In the figure 2.11 there is a finished experiment where the local machine has successfully simulated each of the test cases using the specified indata and displays the desired output below the "Outdata" header. There is also a status message that accompanies each test case if there is something to highlight such as an error during simulation or similar.

Figure 2.12 shows how the results of the experiment can be visualized with tools within Excel.

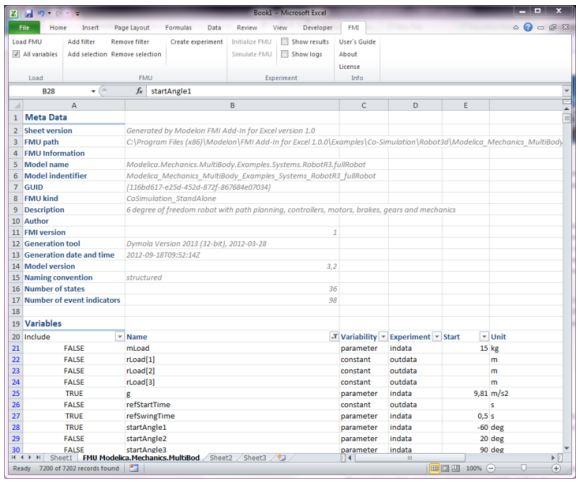

Figure 2.10: *Screen that appears after loading a model.*

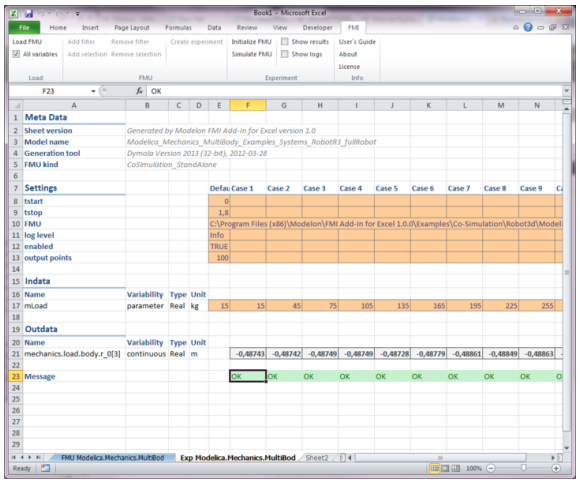

Figure 2.11: *Screen that shows the result after simulating a number of test cases.*

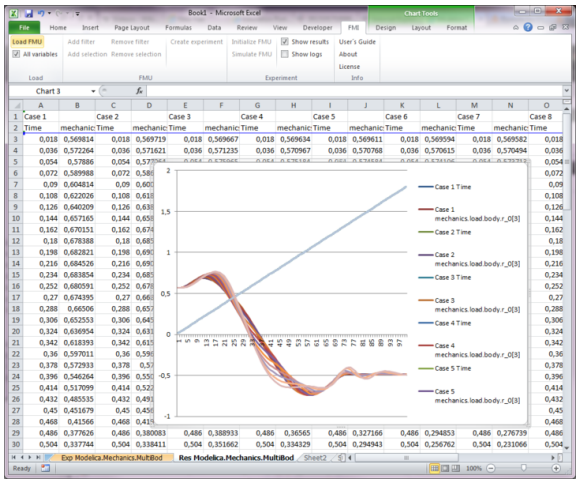

Figure 2.12: *Using Excel tools to visualize data.*

3 Realisation

3.1 Requirements from Modelon

Modelon had a few requirements regarding how the prototype should work. The most significant requirement was for the prototype to utilize cloud services as a means to increase the efficiency of simulations. In addition to the prototype Modelon wanted an investigation of current cloud service providers to get an overview of prices and performance of different solutions. Upon completion the most suitable cloud service provider would be chosen to be part of the prototype.

3.2 Analysis of existing system

The background highlighted several drawbacks with the existing system. These drawbacks can be remedied in several ways. For instance, consider a server that is dedicated to solving complex equations and is able to receive tasks, compute, and then give the result to the client.

Using a dedicated machine is common practice among universities and companies that have lots of people that need to do complex work but do not necessarily have the required computing power in order to get them done in time.

As long as the client's environment is capable of transferring the problem and all of its dependencies to the server while conforming to the protocol that has been established, the client's platform can be anything from a desktop to a mobile device.

A question that may arise is what happens when all the users want to simulate something at the same time. Consider a small company of ten people and a single server that takes care of all the simulations. If each job takes one hour then the users whose jobs are processed last will have to wait a considerable amount of time. This is something that could happen but may be avoided by having several dedicated machines or some kind of priority system in place.

From the users point of view a machine dedicated to simulation will solve the problems since it is possible for users to get simulations done and still use their computer freely.

Having a dedicated machine may be very useful. Setting it up can take some time and the cost may be too big if for instance it doesn't get used as much as it should. Therefore it would be ideal to get the same amount of raw performance but for a limited time. Getting a dedicated machine should be seen as an investment that should pay off over time.

3.3 Prototype design

From the analysis it can be derived that there are multiple components at work here. For instance, the plugin creates experiments and requests for them to be carried out by someone and get the results back. As such the client-server relationship is reasonable in this situation.

However, there are several approaches to the client-server model but also alternative architectural designs such as the peer-to-peer model where participants pool their available resources together and help solve problems for other participants.

There are, of course, different strategies to apply when deciding upon a peer-to-peer architecture. Unlike the client-server model, the peer-to-peer is decentralized, meaning that there is no single node that has more control than any other node in the swarm. In addition, a node can be seen as both the client and the server, or as a consumer and a provider to distinguish from the previously mentioned client-server model. One of the biggest benefits of a peer-to-peer architecture is the robustness

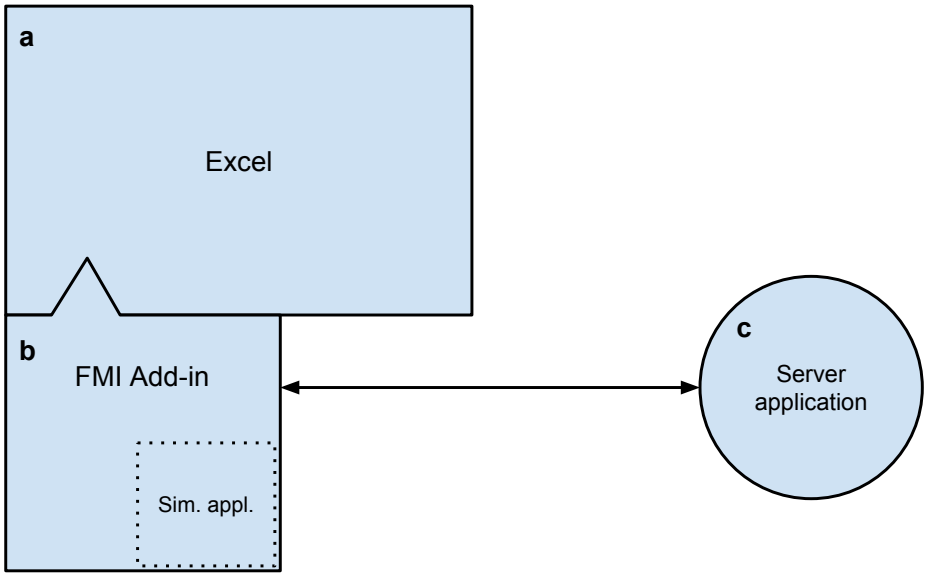

Figure 3.1: *The interaction between the plugin and the server application.*

that comes with the fact that each node can work autonomously and as such several nodes can fail or stop functioning but the swarm will remain intact.

The peer-to-peer architecture remains an interesting alternative and may be up for investigation in the future. However, due to the inclusion of lightweight devices that will use the prototype the net gain in computing power is deemed not enough to keep up with the simulations. But even though the blame falls upon lightweight devices not participating in the swarm as well as a desktop computer a peer-to-peer architecture could still be realized. Using cloud services, a number of virtual machines can be deployed as nodes where these nodes could leverage the lightweight devices with additional processing power.

It would be beneficial if the client did not have to maintain a connection with the server in order to finish the simulation. allowing the client to power down or move to another location (laptop moving across the country) but still have the experiment simulated.

Considering the fact that we are introducing virtual machines to the prototype there has to be some kind of connection with them, either direct or indirect. It is reasonable to ask what happens in a situation if this connection is interrupted. As long as the first request is made successfully there is no point in maintaining a connection just for the sake of keeping it alive. Therefore, a stateless approach such as REST is a suitable alternative.

3.3.1 Plugin

The plugin is a client that is able to construct experiments (collections of test cases) and send them to the server application. While the simulation is in progress the plugin can poll the server application to get an indication on how far along the simulations have come. Once the server application says that everything is done the plugin downloads the result from the server application and feeds it back into the program which formats and displays the result in a spreadsheet.

The plugin behaves similar to how it did before from a users perspective. Just like previously the user begins by selecting a model and specifies any input and output parameters. What the plugin does now is that is sends the entire experiment to the server application. The plugin can poll the server application for a completion percentage. Once the simulation is done the plugin downloads the result and displays

in the spreadsheet.

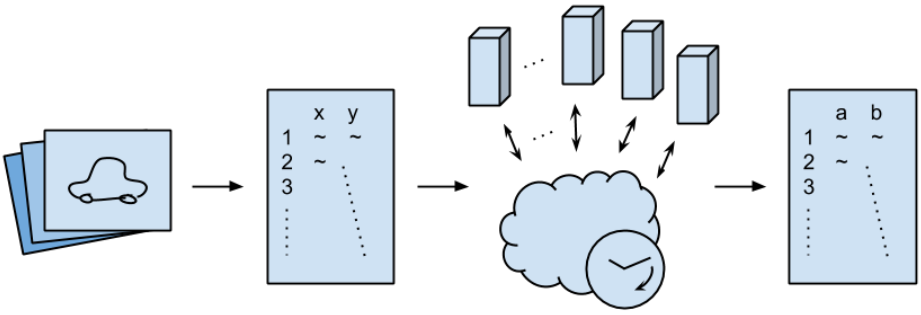

Figure 3.2: *Figure describing the workflow with the prototype.*

3.3.2 Server application

The server application keeps track of all the simulations and makes sure that when someone wants to simulate a model that the job gets done. It communicates with the cloud service provider in order to create the desired number of virtual machines.

The functionality of the API is very limited at this point in time but additional functionality can easily be added.

The server application continuously listens for requests as seen in figure 3.3 (a). Whenever a request appears, the server application parses the contents, figure 3.3 (b). Depending on what kind of request it is, the server application responds accordingly as seen in figure 3.3 (c).

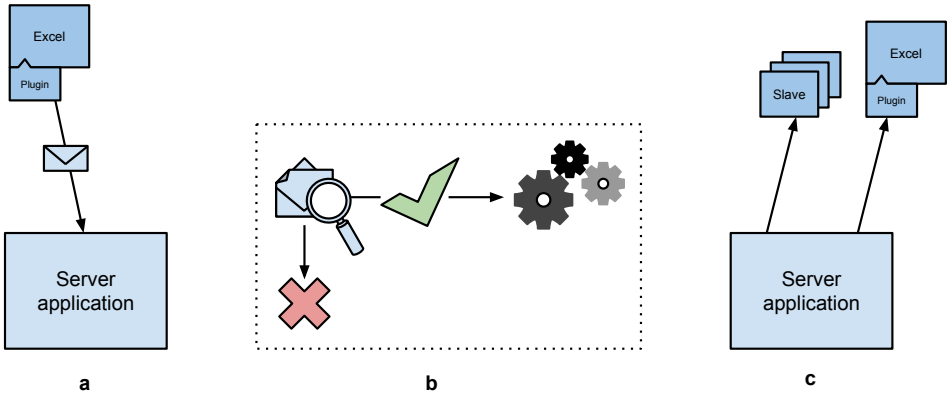

Figure 3.3: *The process of the server application.*

3.3.3 Slave application

Much like the original plugin, the slave application wraps around the simulation application and as such it exists on a Windows virtual machine. This virtual machine is created for the sole purpose of carrying out the simulation. Once the simulation is completed the result is transferred back to the server application.

The virtual machine that host the slave application is either awaken or created depending on how many slaves are available. The experiment is transferred to the slave as seen in figure 3.4 (a). The slave application proceeds to work on each test case producing result files, figure 3.4 (b). Once simulations begin to complete the slave application transfers the result back to the server application, figure 3.4 (c).

When the slave application is done it goes back to sleep.

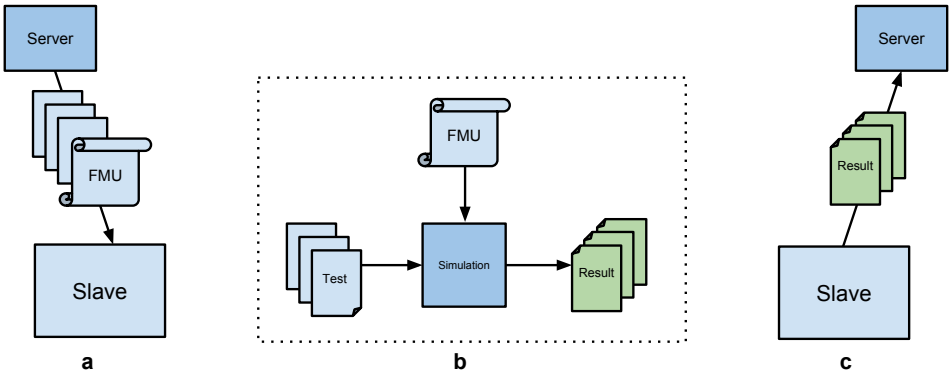

Figure 3.4: *The process of the slave application.*

3.4 Implementation

If we argue that the server application is busy keeping track of all the jobs that are being made by the clients, the server will not have to do any simulation. Instead virtual machines will be used with the sole purpose of computing test cases distributed by the server application.

With cloud services we have the option to create virtual machines on the fly. That is, we can specify how many computers we want to dedicate to simulation. While the server application is keeping track of the jobs it is suitable to let it decide how many virtual machines to use for each job. Therefore the server application is tasked with creating powerful virtual machines, called slaves, with a limited lifetime. Once the time is up the virtual machine is decommissioned and not used any more.

Since the intention is to keep the server application online at all times it would be reasonable to put it on a virtual machine with properties different from the slaves. Although it is possible to use an identical virtual machine, it would be more expensive and the excess resources could be better used. We can let the server application create workers for us with the sole purpose of simulating models; thus minimizing the cost for machines simulating.

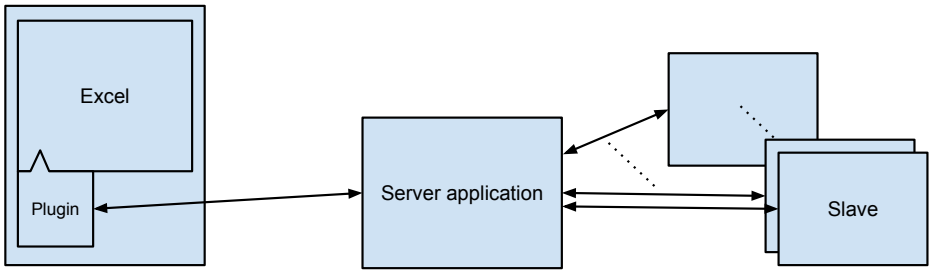

Figure 3.5: *The relationship between the different parts.*

3.4.1 Virtual machines

This prototype makes use of two types of virtual machines. The first is a platform for the server application while the other is the platform for the slave application. What sets them apart depends on the requirements of the applications it is supposed

to run. As mentioned the slave application makes use of a Windows-only binary.

When working with virtual machines it is possible to create a replica, an image, of the state of the virtual machine. Meaning that it is possible to duplicate a virtual machine creating an identical copy (aside from the administrative attributes). This is useful when it comes to the creation of an arbitrary number of slaves. The first virtual machine has to be setup properly in order to obtain a valid image from it but once an image has been created it is possible to use this image as a template for new slaves.

The applications make use of SFTP to transfer files to and from the virtual machines. Out of the box a transfer requires a user with a password in order to transfer a file. In projects where applications communicate to other computers automatically a smoother solution is to use a key file instead of a password.

Customizing an application to use additional services from the cloud service provider couples the application closer with the provider but may translate to less time spent on development.

At the same time as one could benefit greatly from using additional services it could also be the case that it may impact the consumer in the long run. It should be noted that a consumer trading in for a service will have less control over how these services are implemented, price changes, and flexibility.

For both kinds of virtual machines there are a set of directories created to be used with respective application.

The virtual machine that is used for the slave applications run the operating system Windows Server 2008 RC1. The server application, on the other hand, resides on a virtual machine that runs Ubuntu. The motivation behind the different operating systems is mentioned in section 3.4.

In order to receive files from remote locations an SSH server has to be online at the destination machine to manage the incoming connections. Unfortunately there is no application that can function as an SSH server that is natively available in the version of Windows Server 2008 in use here. There is however software available for Windows systems that can act as an SSH server. Many of these programs are proprietary and for them to be used commercially often requires a licensing fee which is not an option as of yet. An alternative solution is to use a port of OpenSSH that is available to Cygwin.

To make sure that the server application is always running there is a scheduled activity that with a few minutes interval run a script that checks if the application is running. If it isn't running it launches the application again.

```
#/bin/sh
# If the process is running do nothing else start application.
if ps -ef | grep -v grep | grep master.jar ; then
    exit 0
else
    /usr/bin/java -jar /path/to/master.jar &
    exit 0
fi
```

where master.jar is the server application packaged as a runnable jar file.

3.4.2 Plugin

The work that was conducted on the FMI Add-in is all new functionality. Being a GUI application some of the new functionality introduced new GUI elements but they are left out since their part in the application is very limited.

The plugin is written in C# and makes use of libraries to facilitate HTTP communication and file transfer.

When the user starts the simulation the application verifies each test case and then sends each file to the server application. Once the files are located on the

server the plugin sends an HTTP POST request to the server application. This request contains all the necessary information and tells the server application to create a new job.

3.4.3 Server application

The server application can send and receive HTTP requests. In this scenario it receives requests from users running the Excel FMI Add-In and will then respond appropriately.

When managing the virtual machines the server application creates HTTP requests that are then sent to the cloud provider's REST API. Through the API, the server application can control the virtual machines' behaviour and the status of the virtual machine (is it running, booting, or provisioning). It is an important part since a virtual machine is not up and running until it explicitly says so and therefore the server application can't communicate with it directly. Managing virtual machines is not instant and can take up to several minutes and is something that the server application is able to take into consideration.

When the server application is distributing work it will transfer the test cases and the models to each slave evenly. For instance, if there are 1000 test cases and 10 virtual machines they will receive 100 test cases each. The test cases and models are transferred using SFTP and once the slave has all the files the server application will initiate a command remotely using SSH that tells the slave to run a jar file with a set of arguments specific to the current job.

The server application consists of essentially a single REST component and that is the job component. The functionality provided is very limited to the prototype itself. It is possible to get information about a job, create a job, and poll the server for status of a given job.

First off, a client sends a POST request to the server application to the URL ../job and the server application parses and verifies the content and responds accordingly. If the request was correct and all files were in place, the server application, would respond with a job id, and the value false otherwise. In a production environment you would most likely want some proper error messages that could indicate to the client what went wrong with the job creation.

Once the job has been created the server application starts as many virtual machines as was specified in the request. Once the virtual machines are up the server application divides the workload among them.

The client is able to get information about jobs by sending a GET request to ../job which will return with a list of information about all jobs. In the same manner if the client wants information about a specific job a GET request is sent to ../job/{id}.

In the current state of the prototype transferral of files is done using the SFTP protocol but a future iteration should migrate this to using the HTTP protocol for the reason to unify the product and remove dependencies.

The server application is written in Java for the same reasons as the slave application with the exception that the requirement regarding native shared libraries does not apply to the server application.

The server application implements SSL/TLS in order to create a secure channel of communication and does not allow any plain HTTP requests.

3.4.4 Slave application

To simulate models Modelon has developed a Windows-specific simulation application that takes a number of arguments and outputs a result file which is an XML file. The simulation application is a wrapper to a library which contains the means to actually perform the calculations.

Being able to simulate is perhaps the most vital component in this project and developing the slave application is a natural first step.

As mentioned in the requirements section the choice of programming language for the slave application was open but that it should be able to simulate models. In order to do so the language had to be able to access functions in the native library.

To make use of the native shared library in the slave application Java Native Access (JNA) is used which makes it possible to map functions in Java to the functions available in the native library. The JNA [twa14] library uses a small native library called foreign function interface library (libffi) to dynamically invoke the native code. With JNA it is possible to load a native library by name and retrieve a pointer to a function within that library. The libffi library is used to invoke functions, without any static bindings, header files, or additional compiling. To make use of a native library (such as the one described above) the developer uses a Java interface to describe functions and structures in the target library, as shown in the example below.

```
public class TestSim {
 public interface SimLibrary extends Library {
   SimLibrary INSTANCE = (SimLibrary)
     Native.loadLibrary("sim", SimLibrary.class);

   int simulate(Object... args);
 }

 public static void main(String[] args) {
   int i = SimLibrary.INSTANCE.simulate(args);
 }
}
```

When it came to implement secure file transfer the decision was to use JSch [Inc06] which is a Java implementation of SSH2. The reasons for this is that JSch is a well regarded library used in many major applications such as Eclipse, Netbeans, and Ant with well-written examples that serve as all the documentation a developer needs in order to get up and running.

While transferring files the application utilizes this specific protocol in order to make sure that the contents sent across the Internet remains unknown to anyone without access.

3.5 Usage

3.5.1 Plugin

The usage of the application hasn't changed in any way other than specifying if any cloud services will be used or not. If so, the plugin will connect to the server application and transfer the test cases and models. A progress bar will appear that shows the status of the job, ranging from status messages of what the server application is doing to a percentage detailing the number of completed simulations.

An addition is that the user has an option here to put the current job into the background meaning that the client is available for other work, such as preparing a new experiment or even turning off the computer. The user can then connect to the server application and fetch the result files later.

3.5.2 Server application

The server application is started with the command:

```
java -jar master.jar
```

and is then idle until any requests arrive. A user interacts with the server application by sending HTTP requests to special URLs. The URLs that the server application responds to are:

../job *GET*
Get information about all jobs.

../job/{id} *GET*
Get information about job with a specific id.

../job *POST* Create job.

3.5.3 Slave application

The application is contained in a single executable JAR file that is started by running the following command:

```
java -jar slave.jar {path to request files}
    {path to result files} {timeout}
    {username} {password}
    {host} {port} {owner} {job id}
```

path to request files *C:\path\to\request\files*
This is the path to a directory containing all the request files, that is an XML files specifying parameters and attributes as well as paths to its log file and model.

path to result files *C:\path\to\result\files*
The path to the output directory, that is where all the result files and the log files are to be stored.

timeout *0.0*
The maximum number of seconds allowed before a simulation is terminated. If 0.0 then let simulations run indefinitely.

The remaining parameters used in transferring files using SFTP and should be self-explanatory to what they are.

username *username*

password *password*

host *127.0.0.1*

port *22*

These parameters are how the server application keeps track of individual jobs that are distributed across a number of slaves. Also used when it is time to transfer the files back to the host. They are used to specify a location on the virtual machine that the server application resides.

owner *owner*

job id *4*

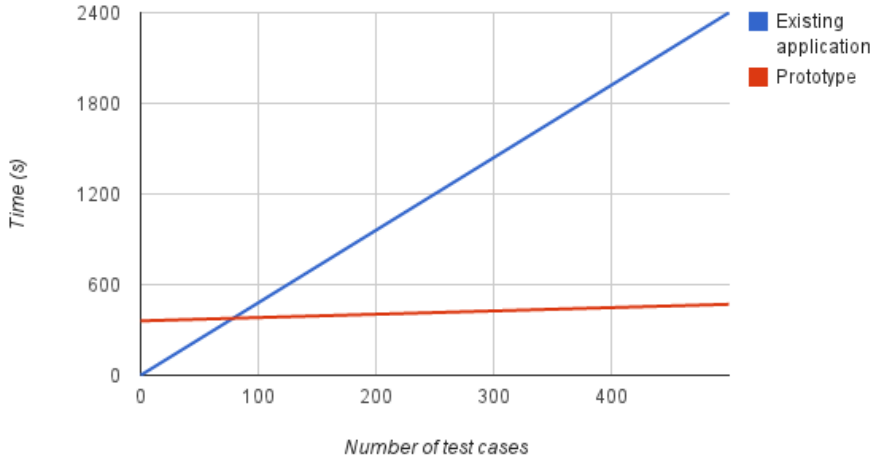

Figure 4.1: *The relationship of the original product and the prototype during a simulation.*

4 Evaluation of results

The development on the prototype has completed its first cycle and can successfully simulate models with the aid of cloud services. The results from a machine where all the simulations have been done locally, and a machine using this prototype match. Thus, this prototype serves as an example of something that is achievable with the aid of virtual machines.

After this iteration, there are some new milestones on the roadmap. Each of these milestones are optimizations that would decrease the amount of time taken from the point of initiating an experiment to the point where the results are displayed in the Excel application. Currently, there are phases in the process that occur sequentially that could run in parallel in order to better make use of the time instead of simply waiting. An example of this is when a user starts an experiment and the Excel Add-in sends a request only when all the files have been transferred to the server application.

A big time sink is the phase where the virtual machines are created and booting which is not instant and can as of writing take up to several minutes which adds a noticeable delay for the user.

Due to these time sinks the prototype is only faster than the original application once the number of test cases increase or the simulation time per test case increase. In other words, if the total time to run an experiment exceeds the flat time it takes to transfer files back and forth in addition to setting up the virtual machines then the prototype is faster. This depends of course on what kind of virtual machines is used and how many that are working concurrently. Creating a single virtual machine with less capacity than the local machine is a waste of time and resources.

In future iterations of this prototype it would be possible to apply a strategy to the virtual machines that are used regarding whether they should shut down once the work has been completed. The strategy would take into account how many people are performing experiments, at what times, budget and so on in order to minimize the downtime.

It is important to note that the prototype suffers from the same limitations as the original application. It is resource intensive, locked to a single platform, and is

just as dependent on the hardware as the original application. The prototype has, however, removed all of these issues from the user's point of view through the usage of the REST API. Even though these issues remain the API enables an increased number of platforms to access it. It allows the user to freely choose what platform to work on with the restriction that it can communicate with the API. At the same time the user is able to allocate his or her resources elsewhere.

Creating a RESTful API was in this case a good solution because the data flows across networks which makes it relatively painless to manage connections. An alternative method would be to use sockets and establish a direct line of communication that way. However, considering that the work to make sure the connections stay up and handling erroneous messages requires more work than what may be useful in this particular scenario. For instance, let's imagine that the prototype uses sockets instead: what would the gain have been? The gain would have been a more responsive prototype since less data is transferred back and forth. HTTP is a verbose protocol and a RESTful architecture would have created a lot of overhead if responsiveness or speed was a requirement.

In future iterations of the prototype it would be a natural thing to implement some kind of push service that automatically notifies the user once an experiment has been completed. It would add to the overall experience when a user can put his or her energy elsewhere and get a notification once the results are back without having to check back to see how it is progressing.

This prototype have used as few cloud provider specific services as possible in order to be as portable as possible. The reason for this is to make it easy to change cloud provider in case of changes with the services or other things that are out of reach for the consumer.

Using cloud services requires the customer to place his or her trust with the provider. This is important in our case since the models are transferred to the cloud provider's virtual machines. This may be an issue for confidential data. The trust requirement isn't limited to this prototype and its models but rather any sensitive information that is hosted by a third party.

5 Conclusion

A question posed in the beginning was whether (and in turn how) it is possible to integrate cloud services into an existing application such as Modelon's FMI Add-in for Excel. The prototype clearly demonstrates an application that successfully makes of use cloud services while being true to the original application.

The method in conjunction with the chosen application designs and architectures work well together and is explained in detail how it is possible to achieve a similar prototype. The technologies used in the prototype works well together partly because of how long these particular technologies have been around and the refinement of applications and libraries that comes with them and partly because of how cloud service providers choose to create their APIs. The cloud provider that was used with this prototype provides a RESTful API, just like the one created in this scenario, which lets the developers use the same mindset.

As of writing, the current implementation of the prototype is faster than the original application once the number of test cases approaches approximately 100. Until up to that amount, it is faster to solve the test cases on the local machine. This is due to how this specific cloud provider creates virtual machines for the prototype which takes most of the time.

That said, there are definitely ways to improve the prototype that would reduce the running time significantly. Since the time spent simulating have decreased as much as it has, local machine versus prototype, in addition to other benefits it is very much reasonable to investigate the possibility to include cloud services for other computationally intensive applications.

A Investigation

A.1 Interacting with a typical cloud provider

Typically, upon registering with a cloud provider, you are taken to the provider's administrative web interface. For instance, Windows Azure calls their "Management Portal". This is a web page where the consumer may interact with the services.

One of the key features of cloud services is that they can be accessed through other applications. This means that if there is a need for some service such checks can appear in an application and doesn't have to be configured beforehand through the administrative interface.

There may be some restrictions regarding the number of services or the capacity they are available at first. For instance, Windows Azure has a cap on the number of virtual machine cores that can be used simultaneously, but this can be lifted.

A.1.1 Infrastructure-as-a-Service providers

Amazon's EC2 was introduced to the general public back in 2006 as a preview version of their virtual computing environment and is a dominant actor on the cloud service provider market with several different types of services [Ser06].

Cluster computing and IBM share a lot of history and today IBM provides their own cloud computing solutions regarding infrastructure, platforms, and applications. Rackspace and HP both provide their cloud computing solutions building upon the open source project OpenStack.

A.1.2 Computing units

When inspecting the properties of different cloud service providers there is a unit that is not apparent to what it represents. Instance types at Amazon [Ser14a] have a different number of EC2 computing units or ECU as they call it. Both Windows Azure and Google have virtual cores as a unit on the description of their respective instance types but Google have defined one virtual core to be equivalent to 2.75 GCEUs (Google computing engine units) [Goo14a].

The reasoning for introducing a unit such as the ECU or GCEU is that it corresponds to a set amount of computing resource that can be used. A company's hardware may not all have identical processors but it is also possible that computer parts change over time, break, and get upgraded as parts get increasingly efficient and cost effective [Ser14b].

A.1.3 Similarities between cloud service providers

With many cloud providers it is possible to interact with their services with the aid of an API. With a cloud provider's API developers can utilize cloud services such as online storage or virtual machines in their applications.

Some providers also have SDKs for a wide range of programming languages. These mostly wrap around the REST API. This particular way of creating APIs enables communication with cloud services for every programming language that is able to send (and receive) messages across HTTP. Since network programming is a fairly common use case there is often suitable libraries for such a task.

Billing options are also similar between the cloud providers. It is often possible to save money in the long run if the usage of services is known since it then would be possible to sign up for a suitable plan.

References

[AB14] D. S. AB. *Dymola*. 2014. URL: http://www.3ds.com/products-services/catia/capabilities/systems-engineering/modelica-systems-simulation/dymola.

[Ass14] M. Association. "Modelica® - A Unified Object-Oriented Language for Systems Modeling. Language Specification". Version 3.3 revision 1. In: (2014). URL: https://www.modelica.org/documents/ModelicaSpec33Revision1.pdf.

[Azu12] W. Azure. *Meet the new Windows Azure*. 2012. URL: http://weblogs.asp.net/scottgu/archive/2012/06/07/meet-the-new-windows-azure.aspx.

[Fie00] R. T. Fielding. "Architectural Styles and the Design of Network-based Software Architectures". 2000. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[Fos02] I. Foster. "What is the Grid? A Three Point Checklist". In: (2002). URL: http://dlib.cs.odu.edu/WhatIsTheGrid.pdf.

[GB12] T. K. George Coulouris Jean Dollimore and G. Blair. *Distributed Systems. Concepts and design*. 2012.

[Goo14a] Google. *Google Compute Engine Pricing*. 2014. URL: https://cloud.google.com/pricing/compute-engine.

[Goo14b] Google. *Google Green*. 2014. URL: www.google.com/green/efficiency.

[IG13] T. IEEE and T. O. Group. *IEEE Std 1003.1*. 2013. URL: http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap03.html#tag_03_19.

[Inc06] J. Inc. *JSch*. 2006. URL: http://www.jcraft.com/jsch/.

[Int13] E. International. *The JSON Data Interchange Format*. 2013. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

[JP99] S. F. John Fanning and S. Parker. *Napster*. 1999. URL: http://en.wikipedia.org/wiki/Napster (visited on 12/17/2014).

[Kre09] K. Krewell. *What's the difference between a CPU and a GPU?* 2009. URL: http://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/.

[Krz03] P. Krzyzanowski. "A taxonomy of distributed systems". In: (2003). URL: http://www.cs.rutgers.edu/~pxk/rutgers/notes/content/01-intro.pdf.

[KS11] A. D. Kshemkalyani and M. Singhal. *Distributed Computing. Principles, Algorithms, and Systems*. Cambridge University Press, 2011.

[LV12] R. P.-C. Lee Badger Tim Grance and J. Voas. "Cloud Computing Synopsis and Recommendations. Recommendations of the National Institute of Standards and Technology". In: (2012). URL: http://www.nist.gov/customcf/get_pdf.cfm?pub_id=911075.

[OE01] M. Otter and H. Elmqvist. "Modelica. Language, Libraries, Tools, Workshop and EU-Project RealSim". In: (2001). URL: https://www.modelica.org/documents/ModelicaOverview14.pdf.

[Pan00] V. Pande. *Folding@home*. 2000. URL: http://folding.stanford.edu/.

[Pro14] M. A. Project. *Functional Mock-up Interface for Model Exchange and Co-Simulation*. Version 2.0. 2014. URL: https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf.

[PW12] P. P. Paolo Costa Matteo Migliavacca and A. L. Wolf. "NaaS: Network-as-a-Service in the Cloud". In: (2012). URL: http://research.microsoft.com/en-us/um/people/pcosta/papers/costa12naas.pdf.

[Rac14] Rackspace. *Rackspace Responsibility*. 2014. URL: http://responsibility.rackspace.com/.

[Ser06] A. W. Services. *Amazon EC2 Beta*. 2006. URL: http://aws.typepad.com/aws/2006/08/amazon_ec2_beta.html.

[Ser14a] A. W. Services. *Amazon EC2 Instances*. 2014. URL: http://aws.amazon.com/ec2/instance-types/.

[Ser14b] A. W. Services. *What is an EC2 Compute Unit and why did you introduce it*. 2014. URL: http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it.

[SN05] J. E. Smith and R. Nair. "The Architecture of Virtual Machines". In: (2005). URL: http://ieeexplore.ieee.org/ielx5/2/30853/01430629.pdf?tp=&arnumber=1430629&isnumber=30853.

[twa14] twall. *Java Native Access (JNA)*. Version 4.1.0. 2014. URL: https://github.com/twall/jna.

[UMB11] C. UMBC. *Is Watson the smartest machine on earth?* 2011. URL: http://www.csee.umbc.edu/2011/02/is-watson-the-smartest-machine-on-earth/.

[W3C08] W3C. *Extensible Markup Language (XML) 1.0 (Fifth edition). W3C Recommendation 26 November 2008*. 2008. URL: http://www.w3.org/TR/2008/REC-xml-20081126/.

[Wik14] Wikipedia. *Plug-in (computing)*. 2014. URL: http://en.wikipedia.org/wiki/Plug-in_(computing) (visited on 12/17/2014).