Global Ecosystem Data Interface

Tari Zahabi

Table of Contents

Abstract

A thesis to propose the possibility of creating an interface to gather data from mycelial networks all around the world. In addition there is a practical part done as a kind of a proof of skill to demo the possibilities, with the description of the process and the technologies used.

Introduction

Since the dawn of time human has had the will to dominate. Our attributes as a species, mainly the ability use tools, and consequently the ability to create complex abstract concepts, has made us able to rise to the top of the food chain globally. The rapid development of technology has made things previously thought as impossible possible, but it has also had its negative side effects. Pollution of natural environments, deterioration of soils and the massive growth of population globally have created a completely new set of challenges for the human kind. Although we are technologically advanced, we still rely on nature to provide us the basics of sustaining life, that is, food, water and oxygen. As a civilization we are at a crossroads, and it's up to us to decide which way to go.

The fact is, Earth has faced many global catastrophies that have almost completely wiped life from the face of Earth. If we intend to survive, it is of utmost importance to gain more knowledge about how life itself works, or we face the same fate as the dinosaurs. There is a limited amount of time available for life on Earth, because in any case the Sun itself will burn itself out, thus creating a massive Supernova that will literally consume our planet and everything on it. There is also the risk of comets, plague, massive volcanic eruptions and even nuclear war. Luckily, human beings are very adept at adapting to changing environments, of which our civilization itself is a proof.

The most revolutionary change in our times has been the invention of Internet, which allows us to transfer information at the speed of light and makes rapid technological development possible. The more data we collect, the more we know about the basic functionalites of the world around us, and when we make this available to everyone, we literally create an extension of our nervous system, a kind of a collective data processing unit. Internet itself reflects nature, and Paul Stamets, a globally respected mycologist, claims that the concept of Internet actually already existed in the form of mycelial networks. He calls mycelial networks the Internet of nature.

To transmit data, a infrastructure is needed, but there also has to be the possibility of processing the data, otherwise its useless. If we think about how we gather knowledge, it could be claimed that reality itself is raw data, which our brains process into usable form, also known as knowledge, or information. This process works through language, which itself is a set of symbols in a certain sequence and associated movements of the muscles of the human body to create sounds associated with the symbols. Actually, we create smaller concepts that we glue together to create new ones. For example, mathematics allows us to write down relationships between symbols that we attach certain values into. This in turn makes possible the abstraction natural phenomena, which leads us into physics. Physics lead to more advancements, and eventually we end up with computers.

First computers had simple operating systems and few functionalities, which we combined to create a new level of abstraction and complexity. This meant that we didn't have to manipulate data on a binary level, instead we mapped the binary

sequences that , for example, draw a pixel on the screen. The sequence of binary was given a label, thus creating another level of abstraction, which basically means that we can just say “draw”, instead of describing every single step in detail. This makes possible the combining of labels, so we can “draw”, “print”, etc.

It is this basic principle of processing and using data that has allowed us to develop as a species. It wouldn't be too far fetched to say that our whole civilization is built on layers of abstraction. We created a simple interface for our basic needs, like supermarkets and shops for clothes, to allow us to specialize and focus more deeply without having to spend time fulfilling our basic needs.

We have to ask where are we going with all of this. The simple answer would be survival. The fact is that as long as we are confined to one planet, we face the risk of total extinction as a species, so the next logical step is to colonize other planets. To make this possible, we need a high level of technology to be able to come up with solutions and resources that we need to accomplish all of this

Mushrooms and mycelia

(http://www.royaltyfreeimages.net/wp-content/uploads/2010/09/royalty-free-images-mushroom-500x375.jpg)

Mushrooms are very important for ecosystems everywhere. They provide the necessary nutrients for other lifeforms and connect plants and animals. They are also able to do complex problem solving and can withstand global catastrophies on a massive scale. They're so capable in this that they can clean up and repair damaged ecosystems, such as Chernobyl, witnessed by Paul Stamets (http://www.youtube.com/watch?v=cwLviP7KaAc, Paul Stamets, The future is Fungi) . They provide natural solutions to diseases like tuberculosis and even cancer. Paul Stamets was able to cure his mother's cancer with the help of mushrooms. They're also able to colonize other planets with spores, which can withstand extreme conditions. Mycelium colonizes soils quite rapidly, and is able to extract minerals from various sources, even stones. To disregard this kind of potential would be a huge mistake.

Mycelium is also in a way aware of everything that happens around it. For example, when animals forage they leave large amounts of loose soil that the mycelium immediately tries to colonize and and use for nutrition. In a way, mycelium is the glue that keeps ecosystems intact and different lifeforms connected.

In a way, mycelium is a infrastructure of transferring nutrients, chemicals and signals. There is variation between different strains, but in general the basic

processes can be mapped and abstracted. This means that with sufficient resources it would be possible to measure the changes and the flow of nutrients in the mycelial network to extract information.

Information is power. Paul Stamets has proof from his research that species which collaborate with mycelium have greater resistance to disease and better availability of nutrients, which is an evolutionary advantage. Mycelium is everywhere, and it's possible to inoculate the places where it's missing. There are various applications of mushrooms, from medicinal purpose to cleaning up polluted environments. Plants communicate with mycelial networks, thus the network holds information about the state of the ecosystem and what's happening inside of it.

With the use of modern technology it should be possible to create an interface for the ecosystem. By measuring different variables from mycelial networks it's possible to monitor vast areas of nature and what's happening inside of them. This could mean that almost every inch of colonizable soil would be under constant surveillance. We would be able to monitor the levels of pollution, the quality of the soil, even the animals and people inside the ecosystem. We already have our own global network, Internet, and combining that with the mycelial networks would mean we'd have access to huge amounts of data previously unavailable. We could monitor different processes and gain deep insights into how mushrooms actually perform the things they do. With the data gathered it would be possible to improve and the tweak the processes even further with the aid of genetical modification and nanotechnology. This in turn would make us able to understand the effects of local phenomena globally, and vice versa. To colonize other planets we first need to know how our planet works, and this could be the first step in accomplishing the human colonization of space.

Let's imagine that we already have done all of this and now we want to colonize a planet. First we need to kickstart the growth of mycelium on the planetary soil, which requires radiation and oxygen. The mycelium would colonize the soil thus making the nutrients necessary for other lifeforms available. With the interface, we could monitor the process in real time, and act accordingly by providing other necessary key species to utilize the nutrients. Eventually this would lead to a self sustaining ecosystem that would be suitable for us humans. Because mycelium is very effective in developing chemicals against viruses and bacteria, it would mean that at the same time we'd have a way to develop medicine to help us battle the bacteria that our immune systems are not used to, thus preventing the risk of contracting deadly diseases when colonizing the planet.

It would be possible to grow food for the whole planetary population, which would mean that the planet would be self sustaining.From the perspective of a interplanetary civilization this would be a great advantage, because this would mean that even though one planet might be destroyed, other planets would be still able to continue and thrive, thus making human civilization very resistant to external threats. Mycelium is also able to withstand almost everything once the planet would be colonized, of which the various global catastrophies are a proof, and thus able to recover.

The process

The process of accomplishing this has to be divided into different levels of abstraction. The first and the highest level is the Mycelial Internet, which means the new network created by connecting Internet and mycelium. The second layer is the interface itself, just as a concept. The third layer is dividing the parts of the interface into objects. The fourth layer is the internal functionalities of the objects, such as input and output of data. The fifth layer is the actual functionality of the parts of the system, or the contents of the objects.

To explain it in a different, let's look at it like we're preparing to make everything ready. This time the order is different. First, we have functioning sensors, because they are the basic building block of the system. They read and possibly write data into the mycelial network. To have a functioning fith layer, the physical parts of the system have to be connected and data has to be able to travel between sensors and the data-gathering server, that shall later be connected to Internet.

Fourth layer requires that the fifth layer is ready. In the fourth layer the basic operations of the object are defined, and have to be functional. The fourth layer doesn't require that the whole object is functional. The operations could be something like this:

- Read/Write
- Debugging mode
- Setup/Configuration

In practice each sensor is an object, and one mycelial network with sensors is another one that contains the sensors-objects. For example, if a sensor object has working setup but doesn't read any data, the network-object can still interact with the object, thus it has enough functionality to complete the fourth layer.

The third layer requires that the objects are completely functional and able to be connected to each other. This means that there's at least one network-object, or a mycelial network with sensors, that is functioning fully.

The second layer is almost the final product, ready to be read data from, but it lacks the user interface. First layer is the final product, with users and remote access with an web interface.

To connect Internet and mycelial networks the first step is to create the necessary infrastructure to transfer data between the two. One possibility is to use existing infrastructure in areas where it is available, such as

telecommunications networks. Every network would have a centralized server that would gather the data from the sensors and pass it further. The server could also pass data to the sensors, which could also send it further on to the mycelial network in case this functionality would be possible and needed. The server itself could be a very simple computer, such as Raspberry Pi, powered by solar energy, thus almost maintenance free. One server could read data from several separate networks, or only just one, depending on the circumstances. The data would be stored on remote servers, and the local server would only pass the information forward without storing anything, thus minimizing the need for electricity and maintenance that comes with handling storage media.

The construction of the system could possibly be accomplished with the use of drones that would deliver the local servers to their locations. Each area would be mapped and the sensors could be shot into the soil from air to their designated locations. The locations could be mapped by looking for mycelial excretions, and if there would be an accurate way of accomplishing this, it would make the complete automation of the process possible.

The distance between each sensor would have to be calculated for each area, because it depends on many local factors, such as elevation and the amount of vegetation.

Each sensor would have an abstract control interface. They could be turned off, the values that they measure could be altered and modified remotely and possibly they could be used to insert data into the mycelial network itself. The possibilities are endless. The interface could be run as a simple webapp with a database backend, so it would be accessible from all around the world.

The question that arises is how to power the sensors. It could be possible to utilize the carbohydrates of the mycelium itself, such as glycogen. Because we share the same protoancestors with fungi, we could utilize some of the same solutions that we use for powering human implants (http://www.sciencedaily.com/releases/2012/06/120613133150.htm) This would make the whole system selfsustainable to a certain extent. The local server could also be equipped with wireless power transmitter that would power the sensors locally, but the server itself would be connected to the electrical grid. Once again the solution depends on the specific circumstances, but the basic idea would be the same in any case.

The drones would be equipped with a gun of some sort, and the sensors would be contained in the bullets. When the sensor is shot into the soil, it would spread it's connectors to reach the mycelium to establish the connection. When ready, it would hibernate until the area was completed, and the local server would send a message to activate the network of sensors. The sensors could be equipped with temporary sources of power, until glycogen or other fuels provided by the mycelium become available. It would mean that if the establishing of the connection failed, it would be possible to find and reuse the sensor.

The sensors themselves would measure different attributes of the mycelium, such as the amount and type of nutrients in circulation. Different levels of enzymes and nutrients should correlate with certain phenomena etc. and these

changes could be transferred into a processable form. The sensors could be equipped with a way to store some of the enzymes, so that they could be released into the mycelium to fine tune or to create certain effects. The difference of measured values between sensors could help locate spots with problems. In the best case scenario the signals of the mycelium could be used to extract exactly the same data that the mycelium itself uses, which would lead to a level of high precision.

Of course all of this is still hypotethical. It might be impossible to insert data into the mycelial network, and the effects of having foreign bodies in a living network of mycelium can actually damage. What really is needed is thorough research on the subject to map the possible ways of interacting with the mycelium, like triggering the defense response of plants artificallly.
(http://onlinelibrary.wiley.com/doi/10.1111/ele.12115/abstract)

On the Internet side of things the requirements aren't that high. Somekind of a database like MariaDB, the necessary infrastructure and hardware and finally the different interfaces for communication, such as SSH and a HTML based website with a graphical interface to the system. Anything special isn't required from the hardware.

Possible applications

(http://www.freestockphotos.biz/stockphoto/9994)

On a smaller scale the technology can be utilized to, for example, create an interface for farmland so that the level of nutrients can be monitored and appropriate amount of fertilizer applied. On a larger scale the flow of nutrients can be seen between different mycelial networks and thus can be compared. All kinds of information can be derived from the collected data, and noticing possible new global phenomena becomes easier.

Mycelium makes the quality of the soil where it grows better. From the point of food production soil-based growing solutions can be made more effective. Real time data can be gathered to be analyzed, and with a simple interface for the system even technologically challenged people can access it. After a longer period of use it would become possible to see the seasonal trends with the data, and it could be gathered automatically for further research.

In practice, he system could be used to automate growing crops and maintaining nutrient levels in the soil. For example, if the soil lacks nitrogen, the system could add more into the soil. Depending on the amount of data and the precision of it, wide areas of land could be guarded by reading from the mycelium if there's someone tresspassing. The biggest problem is that because the idea is still quite new, the possibilites aren't really explored. The wildest ideas suggest the possibility of actually seeing how mycelium creates different chemicals and being able to replicate these processes for the uses of chemical, medicinal and other industries. Most benefits will be reaped by the industry in any case, but eventually there would probably be some consumer level applications, such as specific strains of mushrooms for different plants including the sensors to gather information about the soil and help the gardener optimize his grows.

(http://www.organicgardening.com/learn-and-grow/garden-s-homegrown-ally)

THE PRACTICAL PART

System requirements

The practical part of the thesis was made to be a simple database driven web-app that could be used to demo a very basic webinterface for accessing the data from mycelial networks. It was made using the scripting language Perl, and a web framefork for Perl called Catalyst that is based on the Model-View-Controller idea. The database used is SQLite.

The requirements were the following:

- Draw graphs to demo different possibilites of viewing the data
- Javascript based
- Simple to use and modify
- Easy portability

System design

The system was designed to be simple but effective in what it's doing. The database scheme was designed to hold the example data for the graphs. The data consists mostly of floating point values and text in the database. By using a MVC -based web development framework the possibility of adding more functionality was included in the design by default. The scheme itself is very simple and the tables have no foreign keys or complex data types.

The interface was designed to be minimalist but functional. It features a world map with clickable regions, that show region specific data along with graphs and charts by showing a dialog when a region is clicked. The interface can be easily modified to work on mobile devices, and the device support in general was designed to simple by using only Javascript to do all of the client side scripting.

Implementation

The web application was created with Perl and it's web development framework Catalyst. The database used is SQLite, and on the interface side of thing jQuery and jqPlot were used to do the client side scripting.

Tools

Perl Catalyst

“Catalyst is an open-source Perl MVC web framework that encourages rapid development and clean design without getting in your way by forcing rules.”(http://www.catalystframework.org/)

Perl was chosen because of large and well established user community, because it's multi-platform and because of CPAN (Comprehensive Perl Archive Network). Having a lot of modules ready for use meant that all the boilerplate coding could be minimized, and from the security point of view, the modules are tested and maintained so no need to do everything manually anymore.

A lot of weight was put on available documentation as well, because it just makes everything easier to have things properly documented.

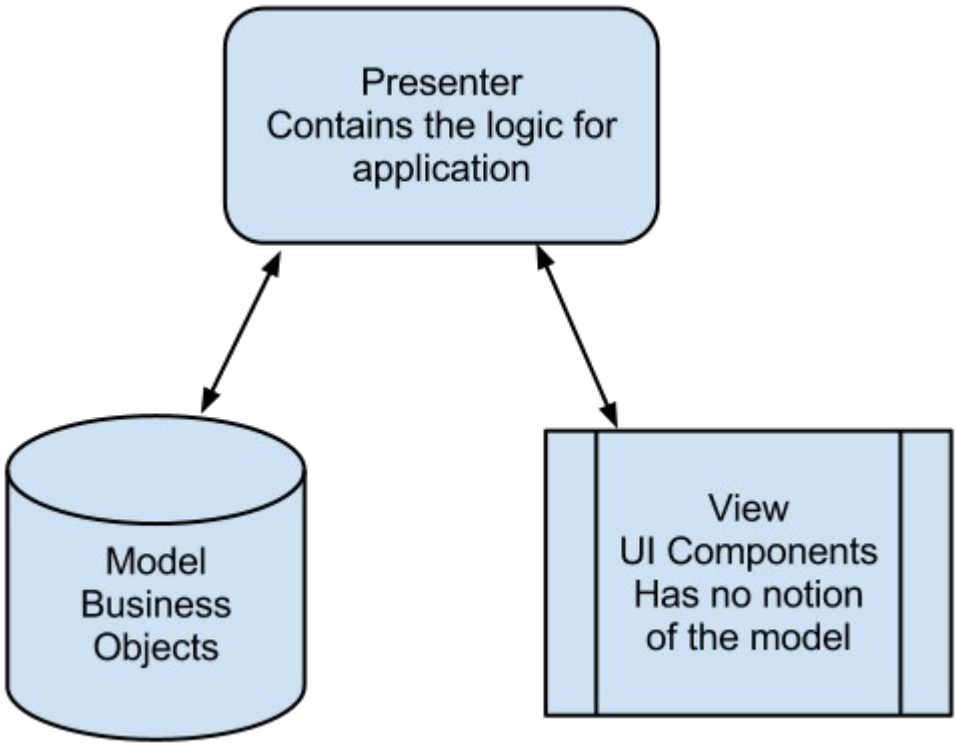
A comparison was made between a few web development frameworks, mainly Mojolicious, Dancer and Catalyst. Catalyst was chosen because of the syntax, a vast amount of tutorials and documentation, and a good selection modules for working with usermanagement and database access, session tracking and all of the basics taken care of quite nicely.

Catalyst is based on the concept of Model-View-Controller. This means that the website or app is divided into three different parts, that are combined and shown as one page when the page is generated. The parts are:

Model	Model can be the data itself, the database or some logic.
View	Defines how the data is laid out on the

	page. Can be for example in HTML.
Controller	The handler for HTTP requests.

(http://blog.codinghorror.com/understanding-model-view-controller/)



(http://upload.wikimedia.org/wikipedia/commons/7/76/Model_View_Presenter.png)

From a development point of view this means that the site is nicely dividable, and the outlook can be separated and worked on independently from all the functionalities. In a larger project it would allow the designer and the developer to focus on their own areas of expertise in a clean and effective way.

Catalyst has a module called Catalyst::View::TT that uses the Template Toolkit (http://www.template-toolkit.org/) to render the view into a page. Creating a basic template that can be modified to user specific needs is automatically created using a script included in the Catalyst package. In every application there are helper scripts included to automate some of the more menial tasks, and creating the database connection can be automated, even with more complex

data-structures and foreign keys involved. The module for accessing the database is called Catalyst::Model::DBI (http://search.cpan.org/~alexp/Catalyst-Model-DBI-0.32/lib/Catalyst/Model/DBI.pm). The module utilizes Object Relational Mapping technology to interface with the database, which in practice means that the database entries are handled as objects, which makes it easier to handle the data, and removes the need to always remember the database scheme.In the Catalyst Wiki (http://wiki.catalystframework.org/wiki/) can be found tutorials and HowTo's for the basic tasks involved in creating a web application, so it's easy even for a beginner to start creating web applications.

SQLite 3

"SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain."

"SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. Think of SQLite not as a replacement for Oracle but as a replacement for fopen() ."

http://www.sqlite.org

Because the webapp is not intented to be used in a commercial setting, but instead to serve as a simple demo of some possibilities, there weren't actually that many requirements, and there will only be a couple of users at maximum. For something like this, SQLite was ideal in it's simplicity and functionality. In the case that the webapp would be deployed into public use, changing the database would be quite easy with the tools provided by Perl Catalyst.

There are some features that make SQLite distinctive:

Zero-Configuration No need for installation or any kind of setup. SQLite works straight out of the box

Serverless No intermediary server process. The database reads and writes data directly from and to the disk.

Only one database file A database used by SQLite is an ordinary disk file. This means easy movability.

Stable Cross-Platform Database File The SQLite fileformat is architechture independent and backwards compatible.

Compact The SQLite library all the options enabled is less than 400KB.

Readable sourcecode Good quality comments and general outlook.

(http://www.sqlite.org/different.html)

SQLite supports SQL, but it has some of it's own commands too for specific tasks. A program called “sqlite3” is used to manage the database file and commit actions within it. It is very quick to prototype database driven applications and backing up the database requires copying only one file. The simplicity of the application also adds to easy maintainability, which frees resources to focus on the content itself, instead of needing to configure and administering a massive database. SQLite also supports up to 100K hits/day, so it can actually handle a moderate amount of traffic before the need to use an enterprise level database arises.

Javascript and Jquery

Using Javascript, or in the case of this project, Jquery, along with Catalyst is fairly straightforward. All the required Javascript libraries can be stored under MyApp/root, from which they can be accessed normally when developing the site itself.

Jquery is a cross-platform library for Javascript that simplifiest the client-side scripting of HTML. It has multi-browser support, which is essential for a website nowadays, clearer syntax than Javascript, support for animations and effects and for plugins. A plugin called jVectorMap was used in the project along with Javascript to create the world map. By using these libraries it's possible to save a lot of time because most of the basic necessities are already taken care of.

Several plugins are available for jQuery, and one called jqPlot was used to draw the graphs on the website with the data retrieved from the database. The world map was implemented using a jQuery plugin called jQVmap (http://jqvmap.com/)

Integrating Javascript on the site is as simple as inserting the code in the wanted template. This also allows inserting values into the code during the processing, through special TemplateToolkit syntax, that is quite clear. In the practical part the graphs aren't updated in realtime, but instead the current values from the database are inserted during the creation of the website when requested. Catalyst takes care of the database connection, returns the database entries as objects and after that it's possible to do whatever with the data. The following is an example of inserting a whole row of data into html:

```
ohlc = [[% FOREACH o IN ohlc %]['[% o.chart_date %]',[% o.a %],[% o.b %],[% o.c %],[% o.d %]],[% END %]];
```

In the example, the " [% FOREACH..." is TemplateToolkit code that goes through an array of objects and inserts the information. In practice the above will look like this:

```
ohlc = [['06/15/2009 16:00:00',136.01,139.5,134.53,139.48],['06/08/2009
16:00:00',143.82,144.56,136.04,136.97],['06/01/2009 16:00:00',136.47,146.4,136,144.67],['05/26/2009
16:00:00',124.76,135.9,124.55,135.81],['05/18/2009 16:00:00',123.73,129.31,121.57,122.5],['05/11/2009
16:00:00',127.37,130.96,119.38,122.42],['05/04/2009 16:00:00',128.24,133.5,126.26,129.19],['04/27/2009
16:00:00',122.9,127.95,122.66,127.24],['04/20/2009 16:00:00',121.73,127.2,118.6,123.9],['04/13/2009
16:00:00',120.01,124.25,115.76,123.42],['04/06/2009 16:00:00',114.94,120,113.28,119.57],['03/30/2009
16:00:00',104.51,116.13,102.61,115.99],['03/23/2009 16:00:00',102.71,109.98,101.75,106.85],['03/16/2009
16:00:00',96.53,103.48,94.18,101.59],['03/09/2009 16:00:00',84.18,97.2,82.57,95.93],['03/02/2009
16:00:00',88.12,92.77,82.33,85.3],['02/23/2009 16:00:00',91.65,92.92,86.51,89.31],['02/17/2009
16:00:00',96.87,97.04,89,91.2],['02/09/2009 16:00:00',100,103,95.77,99.16],['02/02/2009
16:00:00',89.1,100,88.9,99.72],['01/26/2009 16:00:00',88.86,95,88.3,90.13],['01/20/2009
16:00:00',81.93,90,78.2,88.36],['01/12/2009 16:00:00',90.46,90.99,80.05,82.33],['01/05/2009
16:00:00',93.17,97.17,90.04,90.58],['12/29/2008 16:00:00',86.52,91.04,84.72,90.75],['12/22/2008
16:00:00',90.02,90.03,84.55,85.81],['12/15/2008 16:00:00',95.99,96.48,88.02,90],['12/08/2008
16:00:00',97.28,103.6,92.53,98.27],['12/01/2008 16:00:00',91.3,96.23,86.5,94],['11/24/2008
16:00:00',85.21,95.25,84.84,92.67],['11/17/2008 16:00:00',88.48,91.58,79.14,82.58],['11/10/2008
16:00:00',100.17,100.4,86.02,90.24],['11/03/2008 16:00:00',105.93,111.79,95.72,98.24],['10/27/2008
16:00:00',95.07,112.19,91.86,107.59],['10/20/2008 16:00:00',99.78,101.25,90.11,96.38],['10/13/2008
16:00:00',104.55,116.4,85.89,97.4],['10/06/2008 16:00:00',91.96,101.5,85,96.8],['09/29/2008
16:00:00',119.62,119.68,94.65,97.07],['09/22/2008 16:00:00',139.94,140.25,123,128.24],['09/15/2008
16:00:00',142.03,147.69,120.68,140.91],];
```

Notice the last comma, which is left there because after every iteration of the loop a comma is inserted and there is no exception for the last one.

Creating the web application

The practical part of the thesis was begun by researching different available solutions to achieve the requirements. The first step was to choose between different languages. I personally had some background already with Python and Perl and I've been a daily Linux user for years, so one of the requirements was

good support and possible open source background, along with lots of good libraries to use. Because the more lower level languages like C++, Java and some other ones tend not to be preferred in the industry, the language of choice had to be a scripting language.

Although Perl is getting a bit old, it had some characteristics that made it the language of choice. First of all, it is very flexible with syntax, and the Perl slogan actually is “There's more than one way to do it”. This makes it easier for a beginner to focus on the development instead of learning complex syntax. Perl also has very, very sophisticated stringhandling utilities integrated in the language, and it is so expressive that the phenomena of Perl One-liners began. Oneliners can be used from the commandline, and consists only of one line of code, but because Perl is very expressive they can be used to do a lot of data manipulation tasks. A Perl intepreter can also be found on virtually every Unix based operating system, which makes deploying and porting the app easier.

Because Perl is an interpreted language, apps made in Perl are slower than compiled apps. This comes with an advantage of being able to edit the sourcecode without the need of recompiling it. The sacrifice in speed is not big enough to cause trouble, except maybe in the most demanding environments.

CPAN, the Comperehensive Perl Archive Network, contains loads of ready made libraries, code and modules freely available for use. CPAN was definitely one of the biggest pro's when comparing the languages, because in general the documentation and the quality of the code is very good. It also eliminates the need of reinventing the wheel, because especially in the modules related to Catalyst there's a lot of tasks, like session-handling, that are automated and thus leave the developer with free hands to focus on the more complex processes.

When choosing the languages also the available web development frameworks were compared. Catalyst was chosen because of it's use in the industry on large websites, and it is a very powerful tool for professional use. The aforementioned documentation with nice examples included made the use of the framework quite easy from the start, although a great deal of Linux background will definitely prove useful. Everything is done via commandline, and all the files related to the app can be edited with the editor of choice. In every new created application comes included some helper scripts that automate creating the database interface, new views and controllers. Same kind of functionalities could be found also in Web2py, but because Perl was preferred, Catalyst was chosen.

Installing Catalyst is very easy. At the moment the requirements are:

1. perl 5.8.6 or higher
2. Catalyst::Runtime Catalyst::Devel
3. some additional perl modules

Catalyst can be installed without root privileges with Local::lib, and installing it is very straighforward using the CPAN. It automatically configures everything and informs if there are some missing modules or anything else going wrong. All that is needed is a simple command, for example:

```
$cpan Catalyst::Runtime Catalyst::Devel
```

which installs the modules Catalyst::Runtime and Catalyst::Devel.

After installation, a new project can be created with the use of a helper script. This will create a directory structure with only the necessary files and scripts to start building a new app. The layout of a Catalyst application is the following:

Catalyst.pl	A helper script to create the skeleton for the application
Makefile.pl	Needed to automatically find the applications path on the filesystem and helps collect the applications dependencies.
The tests	Tests to probe how the app works
The root	The location for files not related to Catalyst or Perl, like static images.
The configuration file	Allows configuring all of the components used in the app
The Perl modules	The actual project files, like controllers, models and views
Helper scripts	Scripts to run a testing server, creating a new view or a database connection etc.

(http://www.catalystframework.org/calendar/2006/3)

For the uses of this project, the root controller (root.pm) can be utilized to hold all the necessary functionalities. When Catalyst receives and processes a request for a website, it will look for functions,subroutines, or as Catalyst calls them, actions to define the response. The actions types are the following:

Namespace-prefixed(:Local)	Matches URLs beginning with http://localhost:3000/my/controller/foo
Root-level (:Global)	The action is mapped directly to the method name ignoring the controller namespace, which matches http://localhost:3000/bar
Handler behaviour modifiers (:Args)	Adds a match restriction. For example, an action with args(1) would match a /foo/bar/*, and args(0) would match ONLY /foo/bar.

Literal match (:Path)	Adds a match restriction. For example, an action with args(1) would match a /foo/bar/*, and args(0) would match ONLY /foo/bar.
Pattern matching (:Regex and :LocalRegex)	Matches paths that correspond with the defined pattern. :Regex works globally (for example http://localhost:3000/item23/order42) where as :LocalRegex works globally and the namespace is checked first (http://localhost:3000/my/controller/widget23)
Chained actions (:Chained)	Matches parts of the path and allows several actions to take care of the request
Private actions (:Private)	Will not match any URL and can only be called from within Catalyst, and does not respond to an URL request
Default:Path	Called when no other action matches the request
Index : Path : Args (0)	Quite a lot like default except it doesn't take any arguments
Begin : Private	Called after the request has been received and the required controller has been identified, but before any URL matching actions are undertaken.
End : Private	After all URL matching actions have been called, Catalyst will call the function in the controller with the matching URL
Auto: Private	A special action for making chains. Run after begin, but before URL matching. Multiple auto actions can be called

(http://search.cpan.org/~ether/Catalyst-Manual-5.9007/lib/Catalyst/Manual/Intro.pod#Flow_Control)

Because the application is made for demoing purposes only, the deployment of the server isn't that important. None the less, Catalyst provides several ways of

deploying a ready made webapp. The app can be deployed without an external webserver just by using the built-in Catalyst HTTP server. There is also support for nginx, lighttpd and of course Apache on the Linux side. Apache supports both mod_perl and FastCGI, but for most applications either nginx or lighttpd are recommended over Apache because of their light weight.

Conclusion and last words

Because the goal of the practical part of the project was to demonstrate with an simple app the current possibilites of viewing data online in the context of possibly creating a web interface for the sensors in the mycelium. At the moment this kind of technology and data are just dreams, and so we have to settle just for a general idea of how to show data on a website.

The app is a one page site with three different charts. One draws and exponential line, the other one is a pie chart and the third one is a Open-high-low-close-chart (OHLC). They read some invented data from the database and use it to draw the database, and they are modified from the examples shown on jqPlot website (http://www.jqplot.com/tests/) to get the data from the database. Because the emphasis was on development and functionality, there isn't that much on the design side of things. There is an interface called AutoCRUD provided by Catalyst to edit the database entries and thus the graphs. The page can be used to easily show different kinds of data in different forms.

In this project the following things were explored:

- Perl and Catalyst framework (Theory and how to use them)
- HTML,Javascript,jQuery and jqPlot
- TemplateToolkit
- Model – View - Controller (theory)
- SQL and SQLite
- Basic networking theory related to web development

Although the practical part of the project is quite simple, personally it was quite challenging to work with a whole new technology. Catalyst is a very powerful framework and that can sometimes make it very complicated. Finding the data and modules that are up-to-date can be also a feat in itself, because of the sheer amount of tutorials and other information. Completing the project has given me the ability to develop web apps with interactivity and database interaction. It has also given me a wide view of the process of creating, maintaining and understanding websites and web apps. There is no limit to what can be

accomplished with Catalyst and some jQuery, because they both have such a large user base and thus feature a vast amount of different libraries and modules for practically anything. Hopefully in the future the technology to integrate the mycelial networks with such web applications exists, and finally this idea could be actually made reality.

Sourcefiles

root.pm

```
package demo::Controller::Root;
use Moose;
use namespace::autoclean;

BEGIN { extends 'Catalyst::Controller' }
#http://www.scenicreflections.com/files/brown%20mushroom%20Wallpaper__yvt2.jpg
#
# Sets the actions in this controller to be registered with no prefix
# so they function identically to actions created in MyApp.pm
#
__PACKAGE__->config(namespace => '');


__PACKAGE__->config( default_view => 'HTML' );
=encoding utf-8

=head1 NAME

demo::Controller::Root - Root Controller for demo

=head1 DESCRIPTION

[enter your description here]

=head1 METHODS

=head2 index

The root page (/)

=cut

sub index :Path :Args(0) {
    my ( $self, $c ) = @_;
            $c->stash(exponential => [$c->model('DB::Exponential')->all]); #get the data
            $c->stash(pie => [$c->model('DB::pie')->all]);
            $c->stash(ohlc => [$c->model('DB::ohlc')->all]);
            $c->stash(template => 'demo1.tt');
}
```

```
=head2 default

Standard 404 error page

=cut

sub default :Path {
    my ( $self, $c ) = @_;
    $c->response->body( 'Page not found' );
    $c->response->status(404);
}

=head2 end

Attempt to render a view, if needed.

=cut


sub end : ActionClass('RenderView') {}

=head1 AUTHOR

Catalyst developer

=head1 LICENSE

This library is free software. You can redistribute it and/or modify
it under the same terms as Perl itself.

=cut

__PACKAGE__->meta->make_immutable;

1;
```

DB.pm

```
package admin::Model::DB;

use strict;
use base 'Catalyst::Model::DBIC::Schema';

__PACKAGE__->config(
```

```
    schema_class => 'admin::Schema',

    connect_info => {
        dsn => 'dbi:SQLite:test.db',
        user => '',
        password => '',
        on_connect_do => q{PRAGMA foreign_keys = ON},
    }
);

=head1 NAME

admin::Model::DB - Catalyst DBIC Schema Model

=head1 SYNOPSIS

See L<admin>

=head1 DESCRIPTION

L<Catalyst::Model::DBIC::Schema> Model using schema L<admin::Schema>

=head1 GENERATED BY

Catalyst::Helper::Model::DBIC::Schema - 0.62

=head1 AUTHOR

A clever guy

=head1 LICENSE

This library is free software, you can redistribute it and/or modify
it under the same terms as Perl itself.

=cut

1;
```

HTML.pm

```
package FileTreeExample::View::HTML;

use strict;
use base 'Catalyst::View::TT';
```

```
__PACKAGE__->config({
    INCLUDE_PATH => [
        FileTreeExample->path_to( 'root', 'src' ),
        FileTreeExample->path_to( 'root', 'lib' ),
                        FileTreeExample->path_to( 'root', 'src' )
    ],
    PRE_PROCESS  => 'config/main',
    WRAPPER      => 'site/wrapper',
    ERROR        => 'error.tt2',
    TIMER        => 0,
    TEMPLATE_EXTENSION =>'.tt2',
    render_die   => 1,
});

=head1 NAME

FileTreeExample::View::HTML - Catalyst TTSite View

=head1 SYNOPSIS

See L<FileTreeExample>

=head1 DESCRIPTION

Catalyst TTSite View.

=head1 AUTHOR

brandon,,,

=head1 LICENSE

This library is free software. You can redistribute it and/or modify
it under the same terms as Perl itself.

=cut

1;
```

demo1.tt

```
<!DOCTYPE html>
<html>
<head>
```

```
  <title>jVectorMap demo</title>
  <script src="jquery-1.11.1.js"></script>
  <!--[if lt IE 9]><script language="javascript" type="text/javascript" src="excanvas.js"></script><![endif]-->
  <script language="javascript" type="text/javascript" src="jquery.jqplot.min.js"></script>
  <link rel="stylesheet" type="text/css" href="jquery.jqplot.css" />
  <script type="text/javascript" src="examples/jqplot.pieRenderer.min.js"></script>
  <script type="text/javascript" src="examples/jqplot.donutRenderer.min.js"></script>
  <script type="text/javascript" src="examples/jqplot.dateAxisRenderer.min.js"></script>
  <script type="text/javascript" src="examples/jqplot.ohlcRenderer.min.js"></script>
  <script type="text/javascript" src="examples/jqplot.highlighter.min.js"></script>
  <script type="text/javascript" src="jquery-migrate-1.2.1.js"></script>
  <script type="text/javascript" src="jquery-ui-1.10.4.custom.js"></script>
<script type="text/javascript" src="jquery.vmap.js"></script>
<script type="text/javascript" src="jquery.vmap.world.js"></script>
<script type="text/javascript" src="jquery.vmap.sampledata.js"></script>




  <link rel="stylesheet" href="//code.jquery.com/ui/1.10.4/themes/smoothness/jquery-ui.css">
<script>

console.log($);
$(document).ready(function(){



            $( "#dialog" ).dialog({ autoOpen: false });
            $( "#dialog" ).dialog( "option", "height", 500 );

            $( "#dialog" ).dialog( "option", "width", 600 );

            jQuery('#vmap').vectorMap({
    map: 'world_en',
    backgroundColor: null,
    color: '#ffffff',
    hoverOpacity: 0.7,
    selectedColor: '#666666',
    enableZoom: true,
    showTooltip: true,
    values: sample_data,
    scaleColors: ['#C8EEFF', '#006491'],
    normalizeFunction: 'polynomial',
```

```
        onRegionClick: function(element, code, region)
{



      var isOpen = $( "#dialog" ).dialog( "isOpen" );
                              if(isOpen == 1){

                              $( "#dialog" ).dialog( "close" );

                              }
                              else{

                              $( "#dialog" ).dialog( "open" );

                              $( "#dialog" ).dialog( "option", "title", "Statistics for " + region);

                              plot1.replot();
                              plot2.replot();
                              plot3.replot();
                              $("#chart1").hide();
                              $("#chart2").hide();
                              $("#chart3").hide();
                              }

        }
        });


        $("input[type=submit],input[type=assign], a, button").button();
        $("input[type=submit]").click(function(){

        $("#chart1").toggle();


        plot1.replot();

        });

        $("a").click(function(){


        $("#chart2").toggle();
        plot2.replot();
```

```
            });

            $("button").click(function(){

            $("#chart3").toggle();
            plot3.replot();
            });

            $("input[type=assign]").click(function(){

            $("iframe").toggle();

            });



  var popupStatus = 0; // set value




  var chart_data =   [[% FOREACH e IN exponential %][[% e.x_axis %],[% e.y_axis %]],[% END %]];

  var plot1 = jQuery.jqplot('chart1',  [chart_data],
{ title:'Exponential Line',
  axes:{yaxis:{min:-10, max:240}},
  series:[{color:'#5FAB78'}]
});

  ohlc = [[% FOREACH o IN ohlc %]['[% o.chart_date %]',[% o.a %],[% o.b %],[% o.c %],[% o.d %]],[% END %]];

  $("#crud").attr("src","http://localhost:3000/autocrud");


  var data = [[% FOREACH p IN pie %]['[% p.name %]',[% p.percentage %]],[% END %]];

  var plot2 = jQuery.jqplot ('chart2', [data],
    {
      seriesDefaults: {
        // Make this a pie chart.
        renderer: jQuery.jqplot.PieRenderer,
        rendererOptions: {
          // Put data labels on the pie slices.
```

```
        // By default, labels show the percentage of the slice.
        showDataLabels: true
      }
    },
    legend: { show:true, location: 'e' }
  }
);
var plot3 = $.jqplot('chart3',[ohlc],{
  // use the y2 axis on the right of the plot
  //rather than the y axis on the left.
  seriesDefaults:{yaxis:'y2axis'},
  axes: {
    xaxis: {
      renderer:$.jqplot.DateAxisRenderer,
      tickOptions:{formatString:'%b %e'},
      // For date axes, we can specify ticks options as human
      // readable dates.  You should be as specific as possible,
      // however, and include a date and time since some
      // browser treat dates without a time as UTC and some
      // treat dates without time as local time.
      // Generally, if  a time is specified without a time zone,
      // the browser assumes the time zone of the client.
      min: "09-01-2008 16:00",
      max: "06-22-2009 16:00",
      tickInterval: "6 weeks",
    },
    y2axis: {
      tickOptions:{formatString:'$%d'}
    }
  },
  series: [{renderer:$.jqplot.OHLCRenderer}],
  highlighter: {
    show: true,
    showMarker:false,
    tooltipAxes: 'xy',
    yvalues: 4,
    // You can customize the tooltip format string of the highlighter
    // to include any arbitrary text or html and use format string
    // placeholders (%s here) to represent x and y values.
    formatString:'<table class="jqplot-highlighter"> \
    <tr><td>date:</td><td>%s</td></tr> \
    <tr><td>open:</td><td>%s</td></tr> \
    <tr><td>hi:</td><td>%s</td></tr> \
    <tr><td>low:</td><td>%s</td></tr> \
```

```
      <tr><td>close:</td><td>%s</td></tr></table>'
    }
  });




});



</script>

</head>
<center>
<body background="/static/images/mush.jpg">>




<div id="dialog" title="Country Data">
<center><button>OHLC</button>
<input type="submit" value="Exponential" />
<a href="#">Pie Chart</a>
</center>
<div id="chart1" data-height="300px" data-width="500px" style="margin-top:20px; margin-left:20px;"
align="center;"></div>
<div id="chart2" data-height="300px" data-width="500px" style="margin-top:20px; margin-left:20px;"></div>
<div id="chart3" data-height="300px" data-width="500px" style="margin-top:20px; margin-left:20px;"></div>
<br>

<br>
</div>
<div id="vmap" style="width: 800px; height: 600px;"></div>
<input type="assign" value="AutoCRUD" /><br>
<iframe id="crud" src="about:blank" width="600" height="400"></iframe>
</center>
<div id="country_map" style="width: 600px; height: 400px;"></div>

</div>
</body>
</html>
```

Makefile.PL

```
#!/usr/bin/env perl
# IMPORTANT: if you delete this file your app will not work as
# expected.  You have been warned.
```

```
use inc::Module::Install 1.02;
use Module::Install::Catalyst; # Complain loudly if you don't have
                               # Catalyst::Devel installed or haven't said
                               # 'make dist' to create a standalone tarball.

name 'demo';
all_from 'lib/demo.pm';

requires 'Catalyst::Runtime' => '5.90060';
requires 'Catalyst::Plugin::ConfigLoader';
requires 'Catalyst::Plugin::Static::Simple';
requires 'Catalyst::Action::RenderView';
requires 'Catalyst::Plugin::AutoCRUD';
requires 'Moose';
requires 'namespace::autoclean';
requires 'Config::General'; # This should reflect the config file format you've chosen
                 # See Catalyst::Plugin::ConfigLoader for supported formats
test_requires 'Test::More' => '0.88';
catalyst;

install_script glob('script/*.pl');
auto_install;
WriteAll;
```

Database scheme

```
CREATE TABLE exponential(x_axis real, y_axis real);
insert into exponential values
(1,2),
(3,5.12),
(5,13.1),
(7,33.6),
(9,85.9),
(11,219.9);




CREATE TABLE pie(name text PRIMARY KEY, percentage int);

insert into pie values
('Heavy Industry',12),
('Retail',9),
```

('Light Industry',14),
('Out of home', 16),
('Commuting', 7),
('Orientation', 9);

CREATE TABLE ohlc(chart_date text, a real, b real, c real , d real);

insert into ohlc values
('06/15/2009 16:00:00', 136.01, 139.5, 134.53, 139.48),
('06/08/2009 16:00:00', 143.82, 144.56, 136.04, 136.97),
('06/01/2009 16:00:00', 136.47, 146.4, 136, 144.67),
('05/26/2009 16:00:00', 124.76, 135.9, 124.55, 135.81),
('05/18/2009 16:00:00', 123.73, 129.31, 121.57, 122.5),
('05/11/2009 16:00:00', 127.37, 130.96, 119.38, 122.42),
('05/04/2009 16:00:00', 128.24, 133.5, 126.26, 129.19),
('04/27/2009 16:00:00', 122.9, 127.95, 122.66, 127.24),
('04/20/2009 16:00:00', 121.73, 127.2, 118.6, 123.9),
('04/13/2009 16:00:00', 120.01, 124.25, 115.76, 123.42),
('04/06/2009 16:00:00', 114.94, 120, 113.28, 119.57),
('03/30/2009 16:00:00', 104.51, 116.13, 102.61, 115.99),
('03/23/2009 16:00:00', 102.71, 109.98, 101.75, 106.85),
('03/16/2009 16:00:00', 96.53, 103.48, 94.18, 101.59),
('03/09/2009 16:00:00', 84.18, 97.2, 82.57, 95.93),
('03/02/2009 16:00:00', 88.12, 92.77, 82.33, 85.3),
('02/23/2009 16:00:00', 91.65, 92.92, 86.51, 89.31),
('02/17/2009 16:00:00', 96.87, 97.04, 89, 91.2),
('02/09/2009 16:00:00', 100, 103, 95.77, 99.16),
('02/02/2009 16:00:00', 89.1, 100, 88.9, 99.72),
('01/26/2009 16:00:00', 88.86, 95, 88.3, 90.13),
('01/20/2009 16:00:00', 81.93, 90, 78.2, 88.36),
('01/12/2009 16:00:00', 90.46, 90.99, 80.05, 82.33),
('01/05/2009 16:00:00', 93.17, 97.17, 90.04, 90.58),
('12/29/2008 16:00:00', 86.52, 91.04, 84.72, 90.75),
('12/22/2008 16:00:00', 90.02, 90.03, 84.55, 85.81),
('12/15/2008 16:00:00', 95.99, 96.48, 88.02, 90),
('12/08/2008 16:00:00', 97.28, 103.6, 92.53, 98.27),
('12/01/2008 16:00:00', 91.3, 96.23, 86.5, 94),
('11/24/2008 16:00:00', 85.21, 95.25, 84.84, 92.67),
('11/17/2008 16:00:00', 88.48, 91.58, 79.14, 82.58),
('11/10/2008 16:00:00', 100.17, 100.4, 86.02, 90.24),
('11/03/2008 16:00:00', 105.93, 111.79, 95.72, 98.24),
('10/27/2008 16:00:00', 95.07, 112.19, 91.86, 107.59),
('10/20/2008 16:00:00', 99.78, 101.25, 90.11, 96.38),

('10/13/2008 16:00:00', 104.55, 116.4, 85.89, 97.4),
('10/06/2008 16:00:00', 91.96, 101.5, 85, 96.8),
('09/29/2008 16:00:00', 119.62, 119.68, 94.65, 97.07),
('09/22/2008 16:00:00', 139.94, 140.25, 123, 128.24),
('09/15/2008 16:00:00', 142.03, 147.69, 120.68, 140.91);