THE ARCHITECTURE OF AN INTUITIVE SCIENTIFIC WORKFLOW SYSTEM FOR SPATIAL PLANNING

Tiberiu FLORESCU
Ph.D, School of Urban Planning, 'Ion Mincu' University of Architecture and Urban Planning, Bucharest, e-mail: tiberiuflorescu@gmail.com

Cătălin N. SÂRBU
Ph.D, School of Urban Planning, 'Ion Mincu' University of Architecture and Urban Planning, Bucharest, e-mail: sarbu52@yahoo.com

Andrei MITREA
Ph.D, School of Urban Planning, 'Ion Mincu' University of Architecture and Urban Planning, Bucharest, e-mail: andrei.mitrea@uauim.ro

Corina CHIRILĂ
Ph.Dc, School of Urban Planning, 'Ion Mincu' University of Architecture and Urban Planning, Bucharest, e-mail: corina.chirila@yahoo.com

Alin D. CHIŞ
Ph.Dc, School of Geography, University of Bucharest, Bucharest, e-mail: alin.d.chis@gmail.com

Vlad COCHECI
Ph.Dc, School of Geography, Babeş Bolyai University, Cluj-Napoca, e-mail: vladcocheci@gmail.com

Alexandru V. COSTIN
Ph.D, Neoton Design, Cluj, e-mail: neotondesign@gmail.com

Abstract. In recent years, the quantity of open data has multiplied dramatically. This newly found wealth of data has raised a number of issues for any research exercise within the field of spatial planning: underpowered traditional tools and methods, difficulties in tracking data, transparency issues, sharing difficulties and, above all, an erratic workflow. Some new research tools to counter this irksome tendency do exist at the moment, but, unfortunately, we feel that there is still ample room for improvement. We have therefore commenced the development of our own research instrument, based on the Scientific Workflow System concept. This paper lays the foundation for its architecture. Once completed, both the instrument and the resulting data shall be freely available, truthful to the spirit of the open source and open data tradition. We strongly believe that spatial planning professionals and researchers might find it interesting and worthwhile for increasing the quality and speed of their work.

Key words: Scientific workflow system, spatial planning, data science

1. Introduction

Recent years have witnessed a sharp increase in complex spatial planning problems. At the same time, big data has expanded rapidly through recurrent additions of new data sets. Consequently, almost every research exercise must face the rise of vast new bodies of information, which become increasingly difficult to handle.

In principle, such a vast amount of readily available data should pave the way for greater transparency in planning, research and decision-making. Unfortunately, this increased data availability comes with a huge drawback: Data comes in many flavours. Hence, one needs to clean, normalise and filter such data, prior to connecting it with other sets of already processed information. This is most certainly a necessary exercise. However, it subjects the researcher to the arduous task of cleaning and sorting data, which is normally a time consuming, repetitive and often boring task. Moreover, it represents just a fraction of the entire problem that he or she aims to address.

In addition, this new kind of research requires an inter-disciplinary approach: statistics, mathematics, computer science, geography, economics and sociology, urban and regional planning, as well as law and politics, to name just a few. Any researcher involved in such activities therefore needs to work with different tools, each coming with its own files, formats and language. It quickly becomes very difficult to keep track of all the data within one's workspace.

Another problem is the difficulty of verifying someone else's work (Boettiger, 2014). This is a rather serious issue, as it prevents other researchers from delineating a commonly agreed body of knowledge. As Starbuck so aptly puts it: *"Because researchers focus on producing research articles rather than knowledge and because all researchers can claim to have made discoveries, there are no limits to researchers' potential productivity and every researcher can be an unchallenged genius"* (Starbuck, 2006). Research practices such as these result in never ending ambiguity and lack of closure.

Against this background, we wish to become completely transparent in both our workflow and in our results. The next few paragraphs illustrate our efforts in reaching this highly ambitious goal.

1.1. Background

Three years ago, we flirted with the idea of computing the road distance in time between each rural municipality and its closest city, in order to get a more intimate feeling for current accessibility levels in Romania. We figured that the same data could later be used to compute a host of other indices at national level (*cf.* Hoyler *et al.*, 2008; Mennis and Guo, 2009).

To our knowledge, this had never been attempted before. The usual practice was either to work with linear distances or to manually sample some road distances and then extrapolate. Needless to say, both methods work only for small areas and rapidly become unusable as the territory grows in size.

Hence, we started looking for suitable data sets. Surprisingly enough, Romanian authorities have not released any official digital road map of the entire country to date. Every map we could find was either printed or lacked any sort of georeferencing. Using such maps would have implied a tremendous effort of preliminary preparations.

We therefore had to turn our attention towards unofficial data sources. From the very beginning, on-line routing services were out of the question, as this research project went far beyond their intended scope.

Google Maps was our first idea. Unfortunately, the public version has a low daily interrogation limit. We therefore duly moved on to our second choice, which was Open Street Maps (OSM). This was a free and open source solution that proved promising. While using it, we discovered it did have some quality issues, but, all in all, we could manage.

We subsequently downloaded the entire map of Romania. Having a base map at our disposal, we started testing different geographic information systems (GIS), for filtering relevant data. Our first try was QGIS, formerly known as Quantum GIS, an open source GIS working environment. We tried to use its routing extension, but it would not work properly. It often crashed, probably because of the very high data load we imposed upon it. All in all, QGIS is a great tool. We did, however, we did encounter our fair share of problems while working with it. Like other open source projects, it is composed of many parts. This makes it easy to break. For the technically minded, it is usually the Python scripts that fail. Even though this problem would not interfere with the main application, it would result in rendering some extensions unusable.

The OSM file is basically a very large Extensible Markup Language (XML) file, which contains all the items on the map, listed in a non-related manner. The XML file is a human readable file format used for data transactions.

With this new insight, we soon changed perspective completely: We saw the entire data filtering process more like a database query problem. We therefore reframed the entire research exercise, by focusing our full attention upon finding a database solution.

Our choice was PostgreSQL, an SQL Database Management System, since it had the spatial extension PostGIS, which adds spatial data support and functionality. We soon found some on-line tools to import OSM data into the database, as well as instructions to perform a routing between two localities.

It seemed easy enough, so we quickly started working on it. We managed to import the data into the database, but we soon hit another difficulty: The examples we had found were only routing between two given municipalities, so if we wished to calculate the distance between a given municipality and all the rest, we would need two loops to feed in the data. Unfortunately, the SQL language is not built for this, but PostgreSQL provides some extensions which extend the language, thereby allowing loops.

It looked simple enough on paper, but regrettably we did not manage to make it work. This time there were no on-line examples available either, from which to draw our inspiration. Furthermore, our programming skills were somewhat underdeveloped at that time.

We therefore started searching for new ways to do it. One idea quickly came to mind: Accessibility analysis is essentially a form of network analysis. Network analysis provide insights into different relations between nodes (Steiner and Ploder, 2008). However, since network analysis has a multiple level granularity, i.e. node, group or system (*cf.* De Socio, 2010), we should clarify matters at this point, by specifying that we are talking here about system level analysis.

Within such a setting, we would use graphs to calculate the shortest path. In principle, it seemed easy enough. After an intensive research effort, the solution became clearer: We needed to use programming. Quite abruptly however, the next question presented itself: What language to use?

The most accessible programming languages used within scientific communities seemed to be Python, Matlab and R. All of them are held in high esteem, with plenty of libraries readily available. We had previously personally encountered Python and it made a good impression. We had also seen R in action before and we liked it as well. So, we decided to first give Python a try. But, while perusing the Python dedicated literature, we soon found out there was a new kid on the block: Julia.

Simply put, *"Julia is a high-level, high-performance dynamic programming language for technical computing, with syntax that is familiar to users of other technical computing environments"* [http://julialang.org/].

We duly devoted ourselves to the study of this novel programming language. We learned Julia enough to start the project, but we soon reached another hindrance: The graph is built and maintained in memory, and in our case, there was simply too much data that needed storing.

Nonetheless, we quickly found a solution: Work with a database. A new search was on to find a graph database. From the available offer, we spent more time on OrientDB, ArangoDB and Neo4j. Our choice was OrientDB, because it is a multi-model NoSQL database with an SQL-like syntax, which we were already familiar with.

We then imported data from the OSM XML file and built the graph. Next, we implemented the routing algorithm and ran it. It was terribly slow.

By that time however, we were already on a tight schedule. We were striving to present our findings at the 'Lumen Rethinking Social Action Core Values in Practice' International Conference. We had no time to waste upon uncertain results, so we settled for a different approach.

We had by then discovered an OSM routing engine solution. The programme is appropriately named the 'Open Source Routing Machine' (*cf.* Luxen and Vetter, 2011). It has a straightforward use: First, you feed it the OSM XML file, from a Command Line Interface (CLI). The file gets processed and the graph is extracted into a separate file. Second, you start the server, pointing to the graph file location. Afterwards, the application runs as a server, waiting for Hypertext Transfer Protocol (HTTP) routing requests, and returns the time, distance, as well as some additional other information, which we deemed unnecessary for this particular research project.

In simple terms, each routing request must contain a pair of GPS coordinates. For each of the two coordinate points, the server determines the closest point on a road, as well as a route between those points. Remember that we did not have them readily available, as we could not find any official data sets, but we were however able to extract them from the

OSM file. Our newly acquired programming skills were put to good use.

Once we had extracted all municipalities and their corresponding GPS coordinates, getting the fastest route with time and distance was the easy feat. The routing engine turned out to be surprisingly fast.

We then calculated the routing distances between all rural and all urban municipalities. For each rural municipality, we needed to determine the closest urban centre, while ignoring the rest. This is a simple database problem. Since we were already familiar with PostgreSQL, we used it to create the new list.

For visualisation purposes, we used the QGIS platform to build our maps. GIS tools provide a good way to analyse spatial patterns (Clinch and O'Neill, 2009; Hong, 2007). However, we discovered some errors while building the maps: Some rural municipalities were un-routable. While in some instances the database did not contain any road leading to them, in others, the only road was mislabelled as a one-way road. Of the more exotic examples, one urban municipality had its city centre misplaced in a nearby municipality, while the Danube Delta proved completely un-routable, since there was no road connecting it to the mainland. Some of these issues we managed to fix, by using the OSM on-line map, but for many, we simply did not have enough time.

At this point in time, we wished to expand our research by adding population to the equation. Population numbers are officially available on-line, paired to each municipality's name. We also managed to find a list of all Romanian municipalities with their corresponding identifiers, as listed within the Local Administrative Units' Information System (*Sistemul Informatic al Registrului Unitatilor Teritorial-Administrative*/SIRUTA).

We subsequently needed to pair population numbers to the SIRUTA code, because there are more municipalities bearing the same name. Seemingly straightforward at first glance, this proved to be no mean feat, as the data proved to be a complete mess. Differences in spelling were slight, but the computer programme was strict. Removing the resulting mismatches costed us a couple of weeks. It was only then that we could compute the distances and draw the maps.

We were pleased with our results: We had managed to produce maps that reflected a reality unseen until then. By the end of the project we reached a similar conclusion to those expressed by a few other researchers, namely, that network analysis is a viable way to research spatial planning issues (Thirstein *et al.*, 2008; Neuman and Hull, 2009; Vincente *et al.*, 2004; Reyes *et al.*, 2009; Strihan, 2008; Terwal and Boschma, 2009; Rychen and Zimmermann, 2009).

We were acutely aware of the fact that many of our approaches could have been better designed and more efficient. One must keep in mind though that we were dealing with a lot of connected technologies, which require some time to understand more thoroughly. In short, we were somewhat unexperienced, to put it mildly, and we were always pressed by some extraneous time constraints.

We had spent a lot of time on auxiliary matters, such as data mining, cleaning data sets and trying to compute travel distances, while devoting comparatively less time to the actual research. At the same time, we rapidly became overwhelmed with huge quantities of resulting data. Every time we tweaked the process and reran the calculations, we created new data sets. Each and every time, we thought that this would be the final batch. In the end, we became increasingly frustrated by encountering the same kind of problems over and over again.

Unfortunately, this conclusion was not restricted to this particular project. We realised we could do a better job faster, if only we could streamline the research process. The idea grew steadily as each new project commenced, to the point where we started searching for actual ways to do it.

If one is to delve in this kind of research, one needs more discipline, as well as a reliable, transparent and verifiable research system. Within the following sections, we shall devote our attention to a particular breed of such a system.

1.2. Currently available solutions

The Workflow Management System (WFMS) is a concept for an infrastructure dedicated to setting, running and monitoring sequences of workflows or sequences of operations. Its use is most notably visible in software applications that are script based, code written in text files and executed sequentially from top to bottom. The Scientific Workflow System (SWS) is a scientifically focused WFMS, which holds great promise for the future of scientific communities (*cf.* Cohen-Boulakia and Leser, 2011).

There are many applications currently available, which are based on SWS. They all have some features in common, such as a friendly interface, real-time visualisation of results, a simplified process for reutilisation, as well as sharing capabilities and transparency. The problem is however, that these solutions have stemmed from different fields of science, thereby following divergent development paths. Thus, even though they have grown to provide tools for the general use, we feel that some important features are still missing.

We also believe that the needs which drove their development became cemented in some specialised knowhow into the final product. For this reason, we felt increasingly uneasy in using them, as we all had a background in architecture and urban planning as opposed to one in genetics or bioinformatics.

Nonetheless, some interesting applications worth mentioning are the Kepler Project, Orange Data Mining and Apache Taverna. There are many others, but listing all available solutions is beyond the scope of this paper. Suffice it to say we studied many of them and they did not meet all of our needs. Interested readers can consult an extensive list on the ‘Scientific Workflow System’ page on Wikipedia.

The Kepler Project is a free and open source software created in 2002. It was initially designed for supporting environmental data processing, but was later extended to support many other disciplines. The application is written using the Java programming language. The core of the application is provided by Ptolemy II, an actor-oriented modelling system (Ludäscher *et al.*, 2005). This system is a module and component based system with actors and directors. Actors are stand-alone components with input and output ports, while directors handle the orchestration of the workflow. This approach is similar to UNIX pipes and makes the components highly reusable. Kepler is available for Windows, Macintosh (OSX) and Linux.

Orange Data Mining is a free and open source software designed to support data mining and machine learning within the Python scripting language. It is also available for Windows, OSX and Linux. The core of the application is written using the C++ programming language and the user has the option to run workflows via Python scripts or via the provided visual programming interface. The available functionality is split in two, some as part of the core language and the rest as Python scripts (Demšar *et al.*, 2013). This approach makes it easy for users to edit and to tweak the functionality of the scripts, while keeping the core application fast.

Apache Taverna is a free and open source software built in Java. The original target was molecular biology computer simulations and computations. An important trait are workflows that allow piping of external applications, both local and remote (Oinn *et al.*, 2006). Taverna has a modular build, with the core running as a server. Taverna Workbench is the desktop graphical interface, where one can create a workflow without prior knowledge of the language (Li *et al.*, 2008). Workflows are written in 'Simplified conceptual workflow language' (Scufl). Resulting workflows can then be run in four ways: via Taverna Workbench, via command line, via remote execution or via a web browser (OnlineHPC). In addition, the workflows are shared via myExperiment social website and are discoverable from the Taverna Workflow. The software is available for all platforms: Windows, OSX and Linux.

A special case is Wolfram Mathematica, which served as a great inspiration to us. Wolfram Mathematica is a commercial product with a history spanning some 30 years. It was released as a mathematics tool but it surpassed that initial niche with many additional functionalities. It is used by researchers and engineers involved in physics, mathematics and computing. Mathematica provides many tools, including connection to scientific instruments, a notebook interface, a workbench for advanced users, and a curated knowledge base, to name just a few. It uses its own language, the Wolfram Language. It can run as a server and provides a protocol for external applications to interact with its kernel. Wolfram applications can be accessed from a browser via a running server. They can be delivered as a Computable Document Format (CDF) as well, and run via the Wolfram CDF Player, a free tool. It can run on Windows, OSX, Linux and Raspbian. Its main drawback is that it is not a free solution.

A separate discussion can be made about adopting one solution and subsequently making it fit our needs. While possible in theory, we would probably be sailing against the winds, since our requirements go beyond any platform's intended purpose.

A second problem is that we would have to develop a separate branch of coding, which may or may not be accepted into the mainstream. Consequently, the code would eventually become hard to maintain and may even not work well after a while. For these reasons, we feel that a clean approach, starting from clear aims, is the best solution for us. It will obviously take more time to produce a working application than adopting one, but we believe it will pay off on the long run.

1.3. Aims

We recognise that the problem described above is twofold: First, we need a reliable analytical tool for tackling spatial problems. And second, we need robust data sets, which have been previously processed transparently and efficiently, in order to shorten research time. In addition, we strongly believe that this instrument should harbour the possibility of catering for the needs of other researchers as well, in an effort to gain community acceptance.

Therefore, our solution is as follows: We wish to create a single open source work environment, with basic functionalities, and a database structure, for storing and retrieving data. In addition, we strongly consider the possibility of easily building custom functionalities, suitable for different groups of researchers.

At the same time, such an environment should easily connect to other databases. Hence, a subsequent step would then be to host a database to store 'clean', normalised data. This database would be available on-line, ready to be used via the application.

We believe there is no need for everyone to process the same data repeatedly. We therefore wish to concentrate upon sharing already published data, and thereby aiding future projects. In addition to data, we intend to share workflows and templates as well. These have a small footprint, so they can also be added via application updates.

Keeping our experience in mind and drawing inspiration from currently available solutions, we commenced our endeavour from a list of desired characteristics for the work environment. Please bear in mind that they are not listed in any particular order. Hence, our solution ought to touch upon the following aspects:

- Open source: From the very beginning, we are committed to building only upon free and open source technologies, as well as to make our own results available in the same way. We strongly believe that an application survives, becomes successful and subsequently evolves, by supporting a large and active community around it. The open source approach makes it easy for others to discover it and to get involved, as demonstrated by the Linux Developer community (*cf.* Torre, 2008);
- Open data: Our endeavours started when large quantities of data became openly available. Open data has therefore become our main focus. We consider open data both the input and the output of the platform. This includes the ability to import and to export from all kinds of formats;
- Modular: We do not have the necessary time, expertise or willingness to build everything from scratch. For that matter, we are not inventing anything new either. We just want to apply several existing technologies to the field of spatial planning. We intend to build a small platform that works on loosely coupled modules. Loosely coupled implies there is minimum interaction between modules. This makes the platform future-proof, with some modules being upgraded or replaced without affecting other modules. Additional functionality may be gained by building and connecting new modules. In short, failure of a module should not affect the rest.

The platform will, however, be affected by the probable upgrades of the underlying technologies;

- Fast: Since we are talking about processing large quantities of data, it is important to use technologies intended and specifically designed for this purpose;
- Platform independent: Data and workflow should be available on any device connected to the internet. The processing unit should be platform independent;
- Remote computing: Some projects require computations running for large periods of time. For such cases setting up a server is a sensitive choice;
- Distributed computing: Sharing processing power with spare or idle computers, to boost one's computing power;
- Multi-user environment: A researcher may have his personal computer, which does not require more than one user, but we wish to be supportive towards research groups employing a shared workstation or server;
- Teamwork support: Some projects demand teamwork. Large projects usually require one or more research teams, with live updates increasing productivity. Teamwork design requires a multi-user environment support;
- Comprehensive multi-tool environment: The user should be able to produce a research paper, from start to finish, without requiring other applications;
- Intuitive homogenous user interface: Easy to understand and easy to use for newcomers, yet allowing technically minded researchers to tinker with the platform at every level;
- Acquire, curate, store, process and share data: Even though data is readily available, it comes in many formats, from many different sources, which use a staggering number of differing templates. Errors and mistakes in labelling are also a rather common appearance. However, dependable data is paramount to being productive in one's research efforts. Throughout our careers, we came to realise that a disproportionate amount of time is consumed with acquiring, cleaning, and preparing data for future computations;
- Graphical representation: Computer graphics are paramount to working with large quantities of data, as they aid visualisation. This includes maps, graphs, diagrams, tables and charts;
- Text processor: In many cases, the particular text processor used for writing scientific articles is considered to be of minor importance. However, journals have specific template requirements. We therefore intend to provide a simple text processor with template capabilities;
- Share projects and workflow: There is no need for every user to recreate the same project. Similar projects or workflows should be readily available to speed up research time;
- Create and share templates: Templates are a general term that refers to several categories: interface templates, i.e. menus, colours, positions, graphical templates, i.e. map colours/lines/types, charts, graphs, as well as text templates, i.e. display and export.

2. Methodology

The aims listed above should, in principle, cover most of the intended uses of the platform and, at the same time, point to the relevant technologies involved. For each technology, we have researched most of the currently available solutions, and subsequently assessed how well they would work for us. The list is obviously not exhaustive, but it does serve as a pretty exact roadmap.

As mentioned earlier, we are strong supporters of open source and open data. This is a commitment that governs all our practices, including the selection of technologies and instruments. We shall therefore support only open source technology.

2.1. Research strategy

To find the technologies best suited to our purpose, we need to follow three main methodological steps:

1. First, to prepare a comprehensive list with candidate technologies and instruments, after perusing the dedicated literature and studying the various developer communities;
2. Second, to compare technical documentations and user experience, in order to make the final selection;
3. And third, to put the selected technologies to the test.

Needless to say, a comprehensive analysis such as this requires time and a lot of patience. At the time of writing (early 2017), we have progressed quite remarkably along this line. However, it is still early days for us to be able to produce a definitive assessment.

2.2. The search for candidate technologies

With the previous list close at hand, we started the search for candidate technologies. Essentially, we learned how each technology worked and what its intended uses were. As it turned out, some technologies covered more than one aim from our list.

Within the following sections, we shall discuss the available solutions, concentrating upon the arguments for and against their adoption in our project. In cases where we find two or more technological solutions that are equally suited for the job, we weigh in the target audience's ease of adoption and its community support. Another important aspect worth mentioning here is that the chosen technology must not collide with other aims.

2.3. Putting select technologies to the test

The third methodological step requires an additional verification of the selected tools by subjecting them to a test. Hence, an actual test project is to be started by using these tools. We are primarily interested in analysing the following criteria: the modularity of the entire project; the technology's ease of use, mainly for the programming languages involved, but also for the built user interface; and finally, its efficiency, i.e. the speed of data processing.

3. Results and discussion

We believe there are multiple ways to reach the same goal, but we need to concentrate primarily upon our intended audience, *viz.* researchers and professionals working in the field of spatial planning. Hence, the following assessment is based on our own experience. As mentioned earlier, we consistently conducted our analysis along the following lines: The intended use of the technology in question, its ease of use and maintenance, its accessibility for the newcomer, its flexibility in working with other technologies, and, probably equally important, its lack of interference with our stated aims.

3.1. Required technologies

We should first concern ourselves with matters related to coding, as code management is a big issue. The common practice is to use a Version Control System (Milewski, 1997). This system keeps track of your stored work. It is therefore possible to work on several versions of the application at the same time without any interference. It is also worth noting that this is a great tool for teamwork as well.

The most prominent solutions available at present are Mercurial and Git, with Git leading the trend. There are some differences in the way they save files. It is difficult however to choose the 'better' option, as each one is intended for a different class of users. As always, there is some dispute on design issues. Hence, while Git is perceived as faster, at least when working under Linux, Mercurial is regarded as more user friendly.

Both solutions have a distributed model, and the repository is copied on all users' computers. The software is installed on the programmer's computer and the files are hosted locally. One can set up a remote host as well, which is useful for sharing and accessibility. Another solution for sharing is the on-line hosting service, which is usually free for public repositories, but there are some space constraints. However, the text is small in size, so this is not a concern for most users.

After a careful review, we chose the Git version control system as our code, which we would like to host in a public repository on GitHub. The data we produce shall also be available on-line. However, we have not chosen a data hosting solution yet.

Furthermore, it makes little sense to build everything from scratch. We therefore wish to build an application that acts as a platform for existing applications to interact. Modularity is key here, as we intend to create a system that supports a network of loosely coupled modules, where each module is a specifically encapsulated application, designed to allow interaction.

Available solutions are Virtual Machine technology and Container technology. A virtual machine encapsulates an entire Operating System (OS), plus the application and its dependencies. There is an additional management layer between the virtual machine and the underlying OS. This creates overhead, thereby taking up resources. The upside is that it can run any OS on top of any other OS and the separation is complete. On a side note, this technology is widely used for cloud infrastructures.

Container technology is new, stable, light and fast, but still under development (*cf.* Zheng and Thain, 2015). Containers provide a novel way to run Linux applications in an encapsulated environment (Merkel, 2014). The capsule contains the chosen Linux distribution OS without the graphical interface, the application and all the application dependencies. If run on a Linux distribution, the container uses the host kernel. This feature makes it faster than the virtualisation solution, but it can only run Linux applications.

The application is therefore platform and hardware independent, since it will always run in the same environment. We believe the container technology to be the optimum solution for us. We also do not regard the Linux restriction as an issue. Instead, we see it more like a feature.

The most prominent and most advanced container solution available is Docker.

The platform has distributions which run on Windows, OSX and Linux, as well as on cloud infrastructure such as Amazon AWS and Microsoft Azure.

3.2. Enter Julia

Since we are planning to work with big data, we appreciate the advantage of using a language that is offering speed, parallelism and cloud computing straight out of the box. From this perspective, Julia is superior to all the previously mentioned languages. As for language and syntax, it is essentially a mix, borrowing from many popular languages like Python, R, Matlab and others.

Due to these similarities, it is easy for anyone having experience in any of the aforementioned languages to start using it. Furthermore, the language is built with integration in mind, meaning one can integrate code written in other supported languages, i.e. C, Fortran, C++, Python, R and others, inside the Julia code, without any speed penalty.

Julia also supports the new trend of Graphical Processing Unit computing. This means that some computations can be performed on the available graphics card, thereby providing a boost to the overall data processing power. Therefore, our choice of code for the main module is Julia.

Let us now concentrate upon matters related to cross-platform functionality. This discussion usually touches upon two different aspects: on the one hand, it has to address functionality for desktops and servers, while on the other, it should also take into consideration mobile platforms.

However, by making use of emerging web technologies, we can build a single application for both segments. The approach is essentially simple: We first build the platform in a client-server manner. The application will run as a server and the interaction is done via a browser. There are some adjustments needed for the web interface to display content in an optimised way for mobiles and tablets, but the required additional code is trivial, compared to the multiple solutions approach.

Container technology is also the answer for remote computing, an instance where the application is run on a remote server. Here, the communication can be resolved via a Virtual Private Network (VPN).

With respect to distributed computing, we currently have two solutions. One is using the container technology to populate available computers with the required data and applications. The other is to make use of Julia's built-in feature for distributed computing.

The reader should bear in mind, however, that containers are ephemeral. Once closed, all changes are inevitably lost. This is a remarkable feature, as it renders containers practically unbreakable. Thus, in case of trouble, all you have to do is to restart them. Everything then gets back to its original state. However, this also means that data cannot be saved inside containers. There is nonetheless the concept of Volume, which creates a link to the actual storage on the host Operating System. This feature essentially enables saving one's work.

Multi-user environment support can be provided by making use of what is called a 'database management system', or DBMS. The same applies for teamwork support.

An intuitive homogenous user interface is paramount to the success of the application. Here, we have several solutions available. The Julia community is currently working on three different web solutions, which provide a way to build a web interface using Julia code. At the time of writing (early 2017), all are still under development. A fourth solution is to build the interface directly using the available current web application technologies, such as HTML5, CSS3 and JavaScript, as a separate web module.

Another interesting solution provided by the Julia community is a wrapper over a Python library, which communicates with the graphical interface building platform Qt. Qt is not built around web technologies, but provides a full set of tools. Hence, graphical representation of data and text processing can be achieved in three ways: by using the readily available JavaScript libraries, by using Qt or by building it from scratch.

We are a small team. It therefore makes more sense to choose one solution to fit all needs. Client-server architecture with a browser as a client provides that solution. For the desktop version, by using the same display technology we can deliver the view via an Electron Platform, instead of employing a browser.

For the interface, we shall try to use the available Julia modules. Just in case they fail, we shall develop the interface as a separate module, using web technologies.

To acquire, curate, store, process and share data we need to make use of one or more DBMSs. However, for import and export purposes, different file formats must be programmed in. For the DBMS, we have the choice of PostgreSQL or MySQL. Other products are available, but these are the most prominent ones. Both DBMSs provide extensions for working with spatial data, but the PostgreSQL extension, PostGIS, has a longer history, which implies a larger community, able to provide much needed assistance (*cf.* Iacovella, 2012).

Sharing projects, workflow and templates requires the use of an open format, a simple text file. This is a programming feature and does not affect the architecture of the project.

Using PostgreSQL can therefore solve many of our requirements: saving data, authentication, multi-user support, teamwork support, spatial variable storage and spatial functions support.

3.3. The architecture of a SWS implementation

So far we have managed to gather a rather long and sometimes tedious list of technologies that satisfy our needs. We have now reached the stage of assembly, where we devote ourselves to the creation of the final product architecture. Its first sketch appears in Fig. 1.

The application is modular, *viz.* every module is encapsulated into a Docker container. In order to run the containers, we need to have the Docker application installed. With respect to the containers themselves, we need the following minimum requirements: a master container, a display container, a database container, and a storage container.

The master server container encapsulates a Julia server, together with all required dependencies. This container is essentially the brain of the entire operation: It has the power to start and stop other containers. The loading and unloading of additional modules is also handled by the master container.

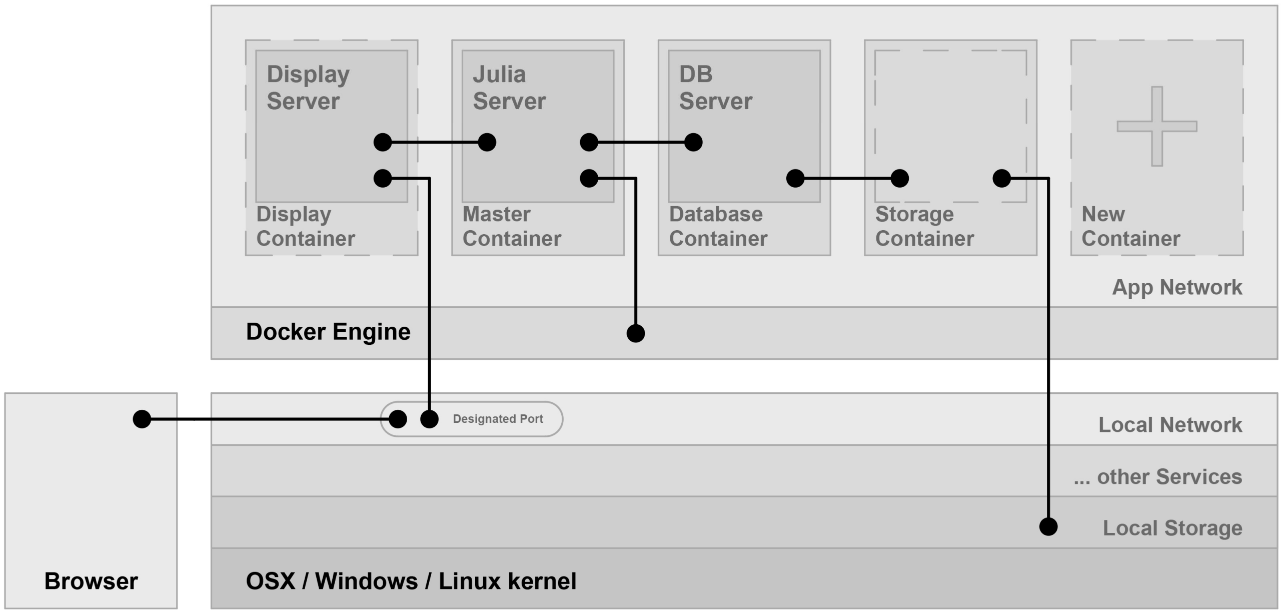

Fig. 1. The architecture of a SWS implementation

The display server handles the creation of the Graphical User Interface (GUI). It is also written in Julia language. The display server takes information from the master container and sends it to the browser. It is also responsible for the interaction with the user. The user interaction events are registered by the browser and sent to the display server. The display server then decides how to respond.

The database server handles the creation, storage and retrieval of data, plus some additional computations. The container encapsulates the PostgreSQL DBMS and its dependencies. It communicates only with the master container.

As mentioned earlier, containers are ephemeral, as they lose all data upon closing. For this reason, we shall create a Storage Container, which can save data to the host's disks.

A display container is not necessary if the computer is not accessed via a browser. This is only the case in distributed computing via containers.

Additional containers can be created at will, in order to add more functionalities. One such container will be the OSRM routing container, mentioned in the introduction.

4. Conclusions

Although many SWS solutions are available, they all come with a target audience. Unfortunately, there is no such solution for the planning field. Moreover, we are reluctant to work upon existing solutions, as they might interfere with our design aims.

In this paper, we gave a very short presentation for some interesting and novel technologies. Some of them can be used to produce a working application with the functionalities that we require.

We are, however, perfectly aware that this is not the only way to proceed forward, and we are certain that we shall face some important design choices along the way. One such example is security. Containers are usually isolated from the platform. In the case of our master container, we need to confer access rights upon it, for it to be able to control other containers. This might pose a security threat, so it needs to be dealt with great care.

Furthermore, Julia is a developing language, which is still changing. Any upgrade might therefore break your code. In addition, there are far fewer available libraries than the languages we have compared it to. However, we still consider it one of the best choices.

The same applies to our choice of PostgreSQL. It is, by all means, a great option, but one should not lose the sight of its market rival, MySQL, which has a larger community and a higher degree of adoption. Nonetheless, we value its independence, as some people frown upon the acquisition of MySQL by Oracle Corporation in recent years. Furthermore, our decision was influenced greatly by PostGIS, which is a well-established extension, boasting an impressive history.

To sum up, we believe that we are on the right track. Based on our newly gained experience, we are confident we can follow up in building the application with the chosen technologies.

REFERENCES

Boettiger C. (2015), *An Introduction to Docker for Reproducible Research, with Examples from the R Environment,* ACM SIGOPS Operating Systems Review, Special Issue on Repeatability and Sharing of Experimental Artifacts **49(1)**: 71-79.

Clinch J. P., O'Neill E. (2009), *Applying Spatial Economics to National Spatial Planning*, Regional Studies **43(2)**: 157-178.

Cohen-Boulakia S., Leser U. (2011), *Search, Adapt, and Reuse: The Future of Scientific Workflows*, ACM SIGMOD Record **40(2)**: 6-16.

De Socio M. (2010), *Marginalization of Sunset Firms in Regime Coalitions: A Social Network Analysis*, Regional Studies **44(2)**, 167-182.

Demšar J., Curk T., Erjavec A., Gorup Č., Hočevar T., Milutinovič M., Možina M., Polajnar M., Toplak M., Starič A., Štajdohar M., Umek L., Žagar L., Žbontar J., Žitnik M., Zupan B. (2013), *Orange: Data Mining Toolbox in Python*, Journal of Machine Learning Research **14**: 2349-2353.

Hong J. (2007), *Firm-specific Effects on Location Decisions of Foreign Direct Investment in China's Logistics Industry*, Regional Studies **41(5)**: 673-683.

Hoyler M., Kloosterman R. C., Sokol M. (2008), *Polycentric Puzzles - Emerging Mega-City Regions Seen through the Lens of Advanced Producer Services*, Regional Studies **42(8)**: 1055-1064.

Iacovella S. (2012), *The Past, Present and Future of GIS: PostGIS 2.0 is Here!*, Linux Journal **2012**: 43-57.

Li P., Castrillo J. I., Velarde G., Wassink I., Soiland-Reyes S., Owen S., Withers D., Oinn T., Pocock M. R., Goble C. A., Oliver S. G., Kell D. B. (2008), *Performing Statistical Analyses on Quantitative Data in Taverna Workflows: An Example Using R and maxdBrowse to Identify Differentially-Expressed Genes from Microarray Data*, BMC Bioinformatics **9**: 334-345.

Ludäscher B., Altintas I., Berkley C., Higgins D., Jaeger E., Jones M., Lee E. A., Tao J., Zhao Y. (2005), *Scientific Workflow Management and the Kepler System*, Concurrency and Computation: Practice and Experience **18(10)**: 1039-1065.

Luxen D., Vetter C. (2011), *Real-time Routing with OpenStreetMap Data*, in: Agrawal D., Cruz I., Jensen C. S., Ofek E., Tanin E. (Eds.), *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 01-04 November 2014, Chicago*, USA, ACM Digital Library, New York, USA, pp. 513-516.

Mennis J., Guo, D. (2009), *Spatial Data Mining and Geographic Knowledge Discovery – An Introduction*, Computers, Environment and Urban Systems **33**: 403-408.

Merkel D. (2014), *Docker: Lightweight Linux Containers for Consistent Development and Deployment*, Linux Journal **2014**: 11-21.

Mileweski B. (1997), *Distributed source control system*, in: Conradi R. (Ed.), *Software Configuration Management*, 18-19 May 1997, Boston, USA, Springer Berlin Heidelberg, Berlin, Germany, pp. 98-107.

Neuman M., Hull A. (2009), *The Futures of the City Region*, Regional Studies **43(6)**: 777-787.

Oinn T., Greenwood M., Addis M., Alpdemir M. N., Ferris J., Glover K., Goble C., Goderis A., Hull D., Marvin D., Li P., Lord P., Pocock M. R., Senger M., Stevens R., Wipat A., Wroe C. (2006), *Taverna: Lessons in*

Creating a Workflow Environment for the Life Sciences, Concurrency and Computation: Practice and Experience **18(10)**: 1067-1100.

Reyes J., Garcia M., Lattimore R. (2009), *The International Economic Order and Trade Architecture*, Spatial Economic Analysis **4(1)**: 73-102.

Rychen F., Zimmermann J. B. (2009), *Clusters in the Global Knowledge Based Economy: Knowledge Gatekeepers and Temporary Proximity*, Regional Studies **42:** 767-776.

Starbuck W. H. (2006), *The Production of Knowledge. The Challenge of Social Science Research,* Oxford University Press, New York, United States.

Steiner M., Ploder M. (2008), *Structure and Strategy within Heterogeneity: Multiple Dimensions of Regional Networking*, Regional Studies **42(6)**: 793-815.

Strihan A. (2008), *A Network-based Approach to Regional Borders: The Case of Belgium*, Regional Studies **42(4)**: 539-554.

Terwal A., Boschma R. (2009), *Applying Social Network Analysis in Economic Geography: Framing Some Key Analytic Issues*, Annals of Regional Sciences **43**: 739-756.

Thirstein A., Lüthi S., Kruse C., Gabi S., Glanzmann L. (2008), *Changing Value Chain of the Swiss Knowledge Economy: Spatial Impact of Intra-firm and Inter-Firm Networks within the Emerging Mega-City Region of Northern Switzerland*, Regional Studies **42(8)**: 1113-1131.

Torre A. (2008), *On the Role Played by Temporary Geographical Proximity in Knowledge Transmission*, Regional Studies **42(6)**: 869-889.

Vincente J., Balland P. A., Brossard O. (2011), *Getting into Networks and Clusters: Evidence from the Midi-Pyrenean Global Navigation Satellite Systems (GNSS) Collaboration Network*, Regional Studies **45(8)**: 1059-1078.

Zheng C., Thain D. (2015), *Integrating Containers into Workflows: A Case Study Using Makeflow, Work Queue, and Docker*, in: Desprez F., Lebre A. (Eds.), *Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing (VTDC '15)*, 15-15 June 2015, Portland, USA, ACM Digital Library, New York, USA, pp. 31-38.

Received: 30 December 2016 • **Revised**: 11 January 2017 • **Accepted**: 17 January 2017

Article distributed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND)