CoeGSS

Centre of excellence

D3.3 – SECOND SPECIFICATION OF NEW METHODS, TOOLS AND MECHANISMS PROPOSED FOR THE SUPPORT OF THE APPLICATION USER AND PROGRAMMER

Grant Agreement	676547
Project Acronym	CoeGSS
Project Title	Centre of Excellence for Global Systems Science
Topic	EINFRA-5-2015
Project website	http://www.coegss-project.eu
Start Date of project	2015-10-01
Duration	36 months
Deliverable due date	2017-06-30
Actual date of submission	2017-08-30
Dissemination level	Public
Nature	Report
Version	2.1
Work Package	WP3
Lead beneficiary	Chalmers
Responsible scientist/administrator	Patrik Jansson
Contributor(s)	Patrik Jansson (editor), Marcin Lawenda, Burak Karaboga, Piotr Dzierżak, Oskar Allerbo, Enrico Ubaldi, Wolfgang Schotte, Cezar Ionescu, Michał Pałka, Eva Richter, Ralf Schneider, Michael Gienger
Internal reviewers	Paweł Wolniewicz at PSNC, Bastian Koller at HLRS
Keywords	HPC, Domain Specific Language, Synthetic Information System, Scalability, Visualisation, Co-design
Total number of pages:	56

The CoeGSS (“Centre of Excellence for Global Systems Science”) project is funded by the European Union. For more information on the project please see the website http:// http://coegss-project.eu/

Version History

	Name	**Partner**	**Date**
V0.1	Patrik Jansson	Chalmers	2017-04-04
V0.2		Chalmers	2017-05-23
V0.3		Chalmers	2017-06-06
V0.9	For internal review 1.1	Chalmers	2017-06-14
Reviewed by	Bastian Koller	HLRS	2017-06-15
V1.0	For internal review 1.2	Chalmers	2017-06-19
Reviewed by	Paweł Wolniewicz	PSNC	2017-06-21
V1.9	For internal review 2	Chalmers	2017-06-26
Reviewed by	Koller and Wolniewicz	HLRS, PSNC	2017-06-28
V2.0	For upload	Chalmers	2017-06-28
V2.1	R2.5 addressed	PSNC, Chalmers, HLRS	2017-08-28
Approved by	Coordinator	UP	2017-09-01

Abstract

Work package 3 (WP3) is a research and development work package with an overall aim to provide a set of tools and methods supporting the work of application programmers and GSS end users. These tools and methods can then be integrated into the portfolio of the centre and, where applicable, as direct services in the CoeGSS portal. This report is a living document, and the release at project month 6 was deliverable D3.2 and the release at month 21 is **deliverable D3.3** of WP3.

The first WP3 deliverable (D3.1) was about the state-of-the-art: methods, tools and mechanisms (MTMs) available off-the-shelf at the start of the CoeGSS project. With this deliverable (D3.3) we capture the status of new MTMs developed and planned by CoeGSS in WP3 (tasks T3.1–T3.6). We start with a description of the CoeGSS workflow and then proceed through the six tasks of WP3 in the same order as in D3.1 and D3.2.

Table of Contents

Table of Abbreviations

ABM	Agent-Based Model
API	Application Programming Interface (a collection of subroutines and tools for building applications)
CKAN	Comprehensive Knowledge Archive Network (a data management system)
CoeGSS	Centre of excellence for Global System Science
D1.4	CoeGSS Deliverable 1.4 detailing the technical situation
D3.1	CoeGSS Deliverable 3.1 on Available MTMs (similarly D3.2, D3.5)
D4.1	CoeGSS Deliverable 4.1 on Pilot Requirements (similarly D4.2)
DoA	Description of Action
DSL	Domain Specific Language
GIS	Geographic Information System
HDFS	Hadoop Distributed File System
HDF5	Hierarchical Data Format 5 (a smart data container)
HLRS	High-Performance Computing Centre Stuttgart (a site in CoeGSS)
HPC	High Performance Computing
JSON	JavaScript Object Notation (open-standard file format)
LAD	Local Authority District (one of the NUTS levels)
M	Month
MS	Milestone
MTMs	methods, tools and mechanisms
NUTS	Nomenclature of Territorial Units for Statistics (an EU geocode standard)
R	Review recommendation
SEDAC	Socioeconomic Data and Applications Center (a NASA data centre)
SIS	Synthetic Information System
SQL	Structured Query Language (DSL designed to manage relational databases)
VR	Virtual Reality
WP	Work Package

List of Figures and Tables

1 Introduction

WP3 is a research and development work package supporting, directly and indirectly, the work of application programmers and GSS end users. As these tools and methods mature, they are integrated into the portfolio of the centre and, where applicable, are made available as direct services in the CoeGSS portal. The overall objectives of this work package for the full three year period are the following according to the Description of Action (slightly edited[1]):

- To propose a prototype version of a heterogeneous environment consisting of HPC infrastructure and cloud storage to be used for scientific use cases (Chapters 2, 3)
- To provide enhanced fault tolerance skills in the proposed architecture (Chapter 3)
- To keep appropriate scalability for future large applications demanding a big data approach by increasing data efficiency (Chapter 4)
- To develop data layer tools and services with a unified interface to the underlying technologies (Chapter 4).
- To provide remote and immersive visualisation (Chapter 5)
- To provide DSLs for assembling GSS simulations (Chapter 6)
- To develop validated numerical methods for GSS simulations (Chapter 7)
- To develop a clear concept and support services for the hardware / software co-design of future needs coming from the users' communities (Chapter 8)

This report is a living document and the release at project month 6 was deliverable D3.2. The second release in month 21 is D3.3 (this deliverable) and the third release in month 31 will be D3.4.

The first deliverable (D3.1) was about the state-of-the-art: methods, tools and mechanisms (MTMs) available off-the-shelf at the start of the CoeGSS project. With D3.2 we proposed new MTMs based on the "gap" between WP3 (research tasks T3.1–T3.6) and WP4 (the pilots). And now, at month 21, we capture a snapshot of the MTMs under development in CoeGSS in this deliverable D3.3 on "CoeGSS Methods".

In CoeGSS the High Performance Computing community (here represented by WP3) meets with the Global Systems Science community (represented by WP4). D3.2 was a first step towards bridging the gap between the two communities and this deliverable captures the progress made in the first 21 months of the project. We start (in Chapter 2) with a description of the common CoeGSS workflow and then proceed through the six tasks of WP3 in the same order as in D3.1 and D3.2.

Note that additional information about the software side has already been described in deliverable D3.5: "Documentation and software on new methods, tools and mechanisms for Release 2 of the Portal" at month 18.

[1] This list is from the WP3 objectives box on page 18 of DoA = Annex 1 of the Grant Agreement (page 101 / 274).

1.1 Executive summary

In Chapter 2 we briefly present the CoeGSS workflow and system architecture (with focus on the differences from D3.2). Each of the six following chapters specify new methods, tools and mechanisms (MTMs) from the point of view of the six tasks of WP3.

- Chapter 3 presents MTMs for scalability in terms of data management and application performance, reliability and monitoring.
- Chapter 4 deals with data management and data analytics. It focuses on storing data and on processing data before the simulations as well as analysing it afterwards. Methods include parallel data analytics, non-relational databases and parameter sweeping.
- Chapter 5 deals with visualisation systems — both remote and immersive. The methods and tools are collected in the CoeGSS Visualisation Toolbox connecting to the CoeGSS portal using COVISE and OpenCOVER. Some example visualisations of the Green Growth pilot are shown to illustrate the methods.
- Chapter 6 describes our work on data representations to allow reuse of synthetic population data, on network reconstruction to build realistic relations between agents from sparse data, on type-based specifications as a common language for reuse of agent-based model components, and on tools for synthetic population generation.
- The focus of Chapter 7 is on ensuring validity and correctness of CoeGSS methods and tools. Methods include interval arithmetics, optimisation algorithms, divide and conquer algorithms, and high assurance software through formalisation using types and functions.
- Chapter 8 is based on the requirements of the pilots when it comes to agent-based simulation on HPC. A graph-based approach to fill the gaps in existing ABM solutions is outlined. The proof of concept for this approach is implemented and evaluated.

Finally, in Chapter **Fehler! Verweisquelle konnte nicht gefunden werden.**, we include an integration plan as requested by the reviewers in the M18 review report and Chapter 10 concludes.

2 Architecture and workflow

The main goal of this chapter is to provide a specific update to the information contained in Chapter 2 of deliverable D3.2. Compared to the information provided there we continued our work towards generalization which will cover all functional blocks present in the GSS processing.

In this chapter the conception of the CoeGSS system architecture and workflow is presented. The architecture defines a conceptual model of the structure, most relevant functional blocks and logical relations between them. Next, the system workflow is discussed which presents a different view (compared to the architecture) and provides information about control and data flow in the CoeGSS system. These two different views allow better understanding the complexity of the entire system.

2.1 Architecture

The architecture diagram is presented in Figure 1: CoeGSS system architecture. In addition to the functional blocks, responsible work packages or tasks are shown in the ellipses. On the very bottom the infrastructure is located represented by the HPC systems, physical data storage and communication devices. Moreover, this block represents also all optimization activities towards performance improving and related with physical infrastructure architecture. Above that, the data management system can be found which in this case is implemented based on the CKAN system. A little bit higher is Synthetic Population Generation responsible for production of profiled synthetic populations.

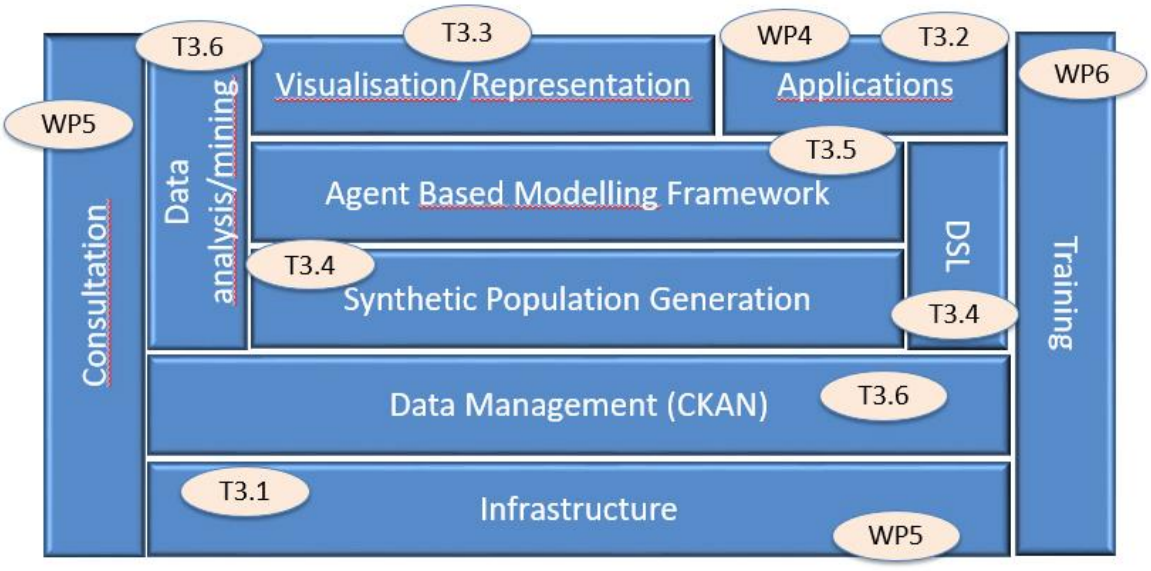

Figure 1: CoeGSS system architecture

The Agent Based Modelling Framework is pledged for simulation of tasks performed according to each model definition. The DSL block illustrates the synthetic population ontology where taxonomy, classes, objects and relations are defined. The data analysis block is composed of pre-

and post-processing applications, which are used for advanced data exploration and manipulation. Visualization concentrates all elements related with data including: advanced and immersive visualization tools, reports generation and presentation. The Applications block represents all programs required by use cases to provide the full functionality of an entire scenario. This is also about methodologies and tools to assess and mitigate uncertainty causes.

All presented blocks are surrounded by no less important services: consultation and training provided respectively by WP5 and WP6. High quality services provided in these two sections are immanent parts of many systems where interaction with users is expected. They are covering all system aspects.

2.2 Workflow

Compared to the architecture, the workflow (Figure 2: CoeGSS system workflow) represents a different point of view on the system. Here, the most important is control and data flow through the most relevant system modules. The presented workflow reflects the general approach for the CoeGSS use cases. Not all blocks are mandatory in all situations. In a given measurement scenario, the workflow can be composed of selected building blocks required by scientific circumstances.

The process starts with data acquisition, which can be harvested in the three ways:

- generated by a synthetic population tool,
- external data source,
- local file prepared by user.

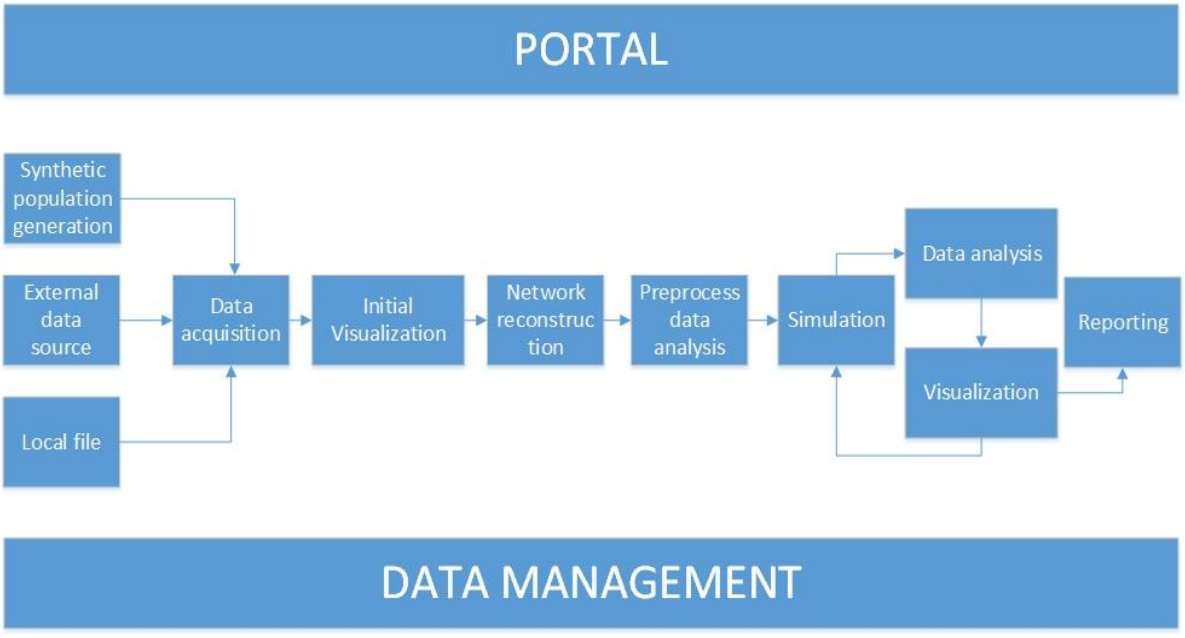

Figure 2: CoeGSS system workflow

The generation of the synthetic population in the dedicated tool can be done either based on micro samples or statistical data provided in advance by the scientist. Both are the seed for production the agent-related information according to the specified parameters.

In the second method data are acquired from the external location where they were prepared by third-party people/software and made publicly available. Examples of such sources are: statistical agency websites or non-profit organizations, which operates databases like MIDAS. More details about the process of incorporating data by CKAN harvester are provided in D3.5 in chapter 4.1.

The CoeGSS system provides also the third method for data supply, which seems to be useful in situation when more sophisticated approach is needed to the synthetic population generation. In this solution information about agents are created manually by researcher and stored in the local file, which can be next imported to the system.

Once data are available in the data management system they can be easily displayed and investigated in preliminary visualization module. It allows the user to get a general overview of available data and to catch most evident bugs. An exemplary screenshot of the preliminary visualization was provided to D3.5 in chapter 4.1.2.

Then, the network reconstruction module is activated which creates relations between agents based on predefined rules. It is required in models where influence of the relations between agents are essential for analysis development. More information about the network reconstruction process can be found in chapter 6.3 of this deliverable.

Once data are complete in the sense of required parameters, relations and error-free data analysis, pre-processing is launched. The analysis task relies on data exploration in the terms of preparation to the simulation process. This process may be of a different nature; one example is rasterization performed in the “Health Habits” use case. More details on pre-processing are provided in deliverable D4.2 chapter 4.2.

In the next step the simulation process is executed using the GSS-specific ABM-framework. It is performed according to the implemented model in order to simulate social behaviours in complex social networks based on agent modelling approach. Bunch of information about used models themselves, their requirements and co-design process are available in D4.1, D4.2 and chapter 8 of this deliverable.

The main goal of the data analysis carried out in the consecutive step on the simulation results is exploration towards finding unrevealed data connotations (more in chapter 4). Moreover, it can be efficiently used as a part of the model calibration process which specification is currently under investigation.

The advanced visualization part is devoted to show the final results of the processing to the user. It must be coupled with simulation tool in the sense of output-input data compatibility. Due to the complexity of this process in terms of computation and data size it is mostly executed on the HPC servers by specialized software like COVISE and only the view is transferred to the user workstation. The chapter 5 provides more knowledge in this respect.

The final part of the workflow is generation of the report from the performed analysis and computation, which in the fact works as the documentation of the work done as well as the topic of further user examination.

3 Enhanced Reliability and Scalability

3.1 Introduction

The chapter describes performed activities towards increasing scalability and reliability in the data management mechanisms. In terms of the computation, scalability achievements in the simulation tool code profiling are presented. Moreover, the progress around monitoring system (as a mechanism to increase the reliability of the entire system) is described.

3.2 Scalability

In the scalability work done so far the focal point was placed on testing and tuning the data management environment in respect of data transfer and data processing within the CKAN server. The data delivered to the system are parsed and converted to the tabular form, which greatly increases the available data manipulation capabilities. Having in mind that system is designed to manage huge amount of data conversion process must be highly efficient. The second focal point is related to increasing scalability of the initial simulation tool (Pandora), specifically when it comes to I/O (input/output) operations. The conducted work and achieved results are presented there.

3.2.1 Data management scalability

In the chapter 3.4 of the deliverable D3.2 the CKAN system is described as primary solution for the data management in the CoeGSS project. In this section we continue our analysis and tests towards obtaining more efficient and scalable solution.

CKAN is a powerful data management system that makes data **accessible** – by providing tools to streamline **publishing**, sharing, **finding** and **using** data. CKAN is aimed at data publishers (national and regional governments, companies and organizations) willing to make their data open and available. The CKAN platform has a web interface and a REST API.

All of a CKAN website's core functionality (everything that can be done with the web interface and more) can be used by external code that calls the CKAN API. The CKAN Client was implemented in the Perl language and it uses the API. The following functionality is available:

- Getting lists of a site's datasets, groups or other CKAN objects,
- Getting a CSV or JSON representation of a dataset, resource or other object,
- Searching for packages or resources matching a query,
- Creating, updating and deleting datasets, resources and other objects,
- Getting an activity stream of recently changed datasets on a site.

A good way to transfer data to the CKAN platform is to use a CKAN Perl Client. Through the client it is possible to up- and download data files (e.g. CSV, TXT, XLS), insert, get and filter records. Depending on the number of records the operation can take several minutes or more (see Table

1). After importing the data into the CKAN DataStore the file from an external source can be deleted, the data remains in the database.

FILE SIZE / LINES	UPLOAD HTTP	UPLOAD HTTPS	PROCESSING BY CKAN	UPLOAD CSV BY 10000 LINES	DOWNLOAD [s]
8 KB / 84 L	1s	1s	0m 1s	0m 4s	1
20 MB / 167553 L	2s	2s	5m 12s	3m 40s	1
74 MB / 640261 L	4s	5s	12m 42s	12m 49s	1
115 MB / 765022 L	6s	6s	16m 00s	17m 29s	2
199 MB / 1336855 L	8s	9s	34m 45s	39m 31s	3
255 MB / 1751555 L	14s	13s	40m 52s	44m 08s	3

Table 1: CKAN Client up- and download tests. Client in Czestochowa (Poland) – server at PSNC

Table 1 shows the up- and download of the CSV file test results of the CKAN platform installed at PCSS. The CKAN Client was installed in Czestochowa University of Technology and made data transfers with server instance at PSNC.

FILE SIZE / LINES	UPLOAD HTTP	UPLOAD HTTPS	PROCESSING BY CKAN	UPLOAD CSV BY 10000 LINES	DOWNLOAD [s]
74 MB / 640261 L	4s	5s	12m 42s	12m 49s	1

FILTER RECORDS [%]	LINES	SAVE CSV TIME [s]
10	64026	22s
20	128052	45s
30	192078	1m 7s
40	256104	1m 29s
50	320130	1m 49s
60	384156	2m 15s
70	448182	2m 30s
80	512208	2m 50s
90	576234	3m 21s
100	640261	3m 28s

Table 2: CKAN Client – filter records and save output to a CSV file

Table 2 shows the test results about filtering and saving output data to CSV files. It can be observed that operations across the input file are four times faster than saving records to the CKAN platform.

The CKAN platform supports a number of extensions. The most important of them are DataStore, FileStore and Extrafields.

The CKAN DataStore extension provides an ad hoc database for storage of structured data from the CKAN resources. Data can be extracted of resource file(s) and stored in the DataStore.

CKAN's FileStore extension allows users to upload data files to the CKAN resources, and to upload logo images for groups and organizations. User will see an upload button while creating or updating a resource, group or organization.

The DataStore is distinct but complementary to the FileStore. In contrast to the FileStore which provides 'blob' storage of whole files with no way to access or query parts of that file, the DataStore is a database alike in which individual datasets are accessible and queryable. To illustrate this distinction, consider storing a spreadsheet file like a CSV or Excel document. In the FileStore this file would be stored directly. To access it the whole file would be downloaded. But if the spreadsheet data are stored in the DataStore, one would be able to access individual spreadsheet rows via a simple web API and make queries over the spreadsheet contents.

3.2.2 Computation scalability

In this section changes and improvements in the Pandora Serializer class are described.

Since the performance of the original Pandora version hosted at GitHub was not satisfying, in view of the targeted HPC usage of the program, some changes in the Pandora Serializer class were done to fix this issue. In Pandora the class Serializer is responsible for the parallel output of result data, residing on agent and raster entities, to disk. This is done by wrapping calls to the parallel HDF5 library's c-API by C++ constructs. In this chapter, we report the changes in the implementation of the Serializer class as well as the runtime improvements resulting from these changes.

The analysis of the original implementation was performed by means of the Cray Performance Analysis Tools (CrayPAT) and the Cray MPI-IO debugging tools. The analyses with CrayPat revealed that the initial implementation spend most of its runtime in routines related to the I/O and the usage of HDF function calls[2]. Due to the HDF5 implementation, based on MPI-IO as the backend that executes the actual write calls, a further analysis by means of the Cray MPI-IO debugging tools was done. This analysis showed that the bad I/O performance was due to massive use of independent write calls[3].

Since it is, even with current parallel file systems like the Lustre file systems connected to the HLRS Hazel Hen system, not possible to grant independent write access to tens of thousands of processes, the CRAY XC40 system uses a tailored MPI implementation that is based on a hierarchical scheme of network communication to so called aggregators. This means whenever a write call is issued by one of the MPI-processes the data are communicated via the network to aggregator processes that in turn are responsible to execute the actual write statement.

To make the aggregator mechanism work, the calls to parallel write statements have to be collective. To achieve this in HDF5 the originally independent file properties were set to collective ones by changing

```
H5Pset_dxpl_mpio(propertyListId, H5FD_MPIO_INDEPENDENT);
```

to

```
H5Pset_dxpl_mpio(propertyListId, H5FD_MPIO_COLLECTIVE);
```

[2] For a detailed guideline on how to use CrayPat please see http://docs.cray.com and http://wiki.coegss.eu/doku.php?id=hazelhen_performance_engineering#craypat.

[3] For a detailed guideline how to use the Cray MPI-IO debugging tools please see http://docs.cray.com http://wiki.coegss.eu/doku.php?id=hazelhen_performance_engineering#analysing_mpi-io.

The effect of this change can be seen in Figure 3 for two examples, executing the random walker example of Pandora on a grid with 5632 x 5632 Cells with 10 million agents for 10 steps with the serializer resolution set to 1 and 10 respectively. Setting the serializer to 1 means writing results in every step, where setting it to 10 means writing the current properties of the agents and the rasters every 10th step.

It can be seen, that the improvement clearly depends on the output amount and the number of MPI processes. While it is possible to reduce the complete problem runtime in the case of 1024 MPI processes and a serializer resolution of 1, by 70% from 3min:49sec to 1min:07sec, it is no improvement visible for a serializer resolution of 10 executed by 64 or 121 MPI processes. For 1024 MPI processes with a serializer resolution of 10 one can again observe an improvement of 23%, which in total numbers, reduced the overall walltime from 42sec to 32sec.

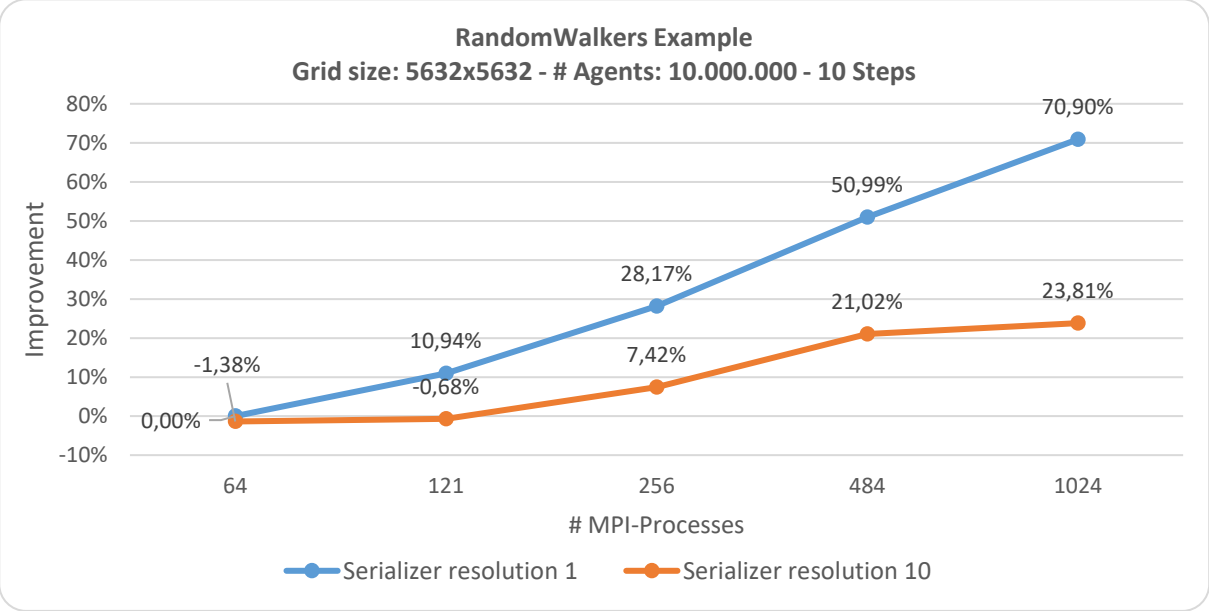

Figure 3: Improvements in total execution time by collective HDF5 write operations

3.3 Reliability

This chapter describes proposed mechanisms to increase reliability in area of the data management system as well as the most vital services by using dedicated monitoring system.

3.3.1 Data management reliability

The CKAN platform can be configured as a high availability cluster. This functionality should work with all recent versions of the CKAN server. To provide redundancy the frontend (CKAN portal) and backend components (PostgreSQL database and Solr platform) should be duplicated. A functional diagram of the redundancy conception is presented in Figure 4.

3.3.1.1 Frontend redundancy

Redundancy on the frontend can be provided by having multiple web servers, each with their own copy of the CKAN code. As CKAN is a WSGI (Web Server Gateway Interface) app any suitable web server setup can be used, but generally, Apache with mod_wsgi is recommended.

A load balancer is then placed in front of the web servers (the nginx server is recommended). DNS servers should be configured to return all the IP addresses of the load balancers.

The frontend needs also a high availability proxy mechanism and memcache service – to keep the CKAN portal login data. These data should be written and read to multiple memcache instances via a high availability proxy service.

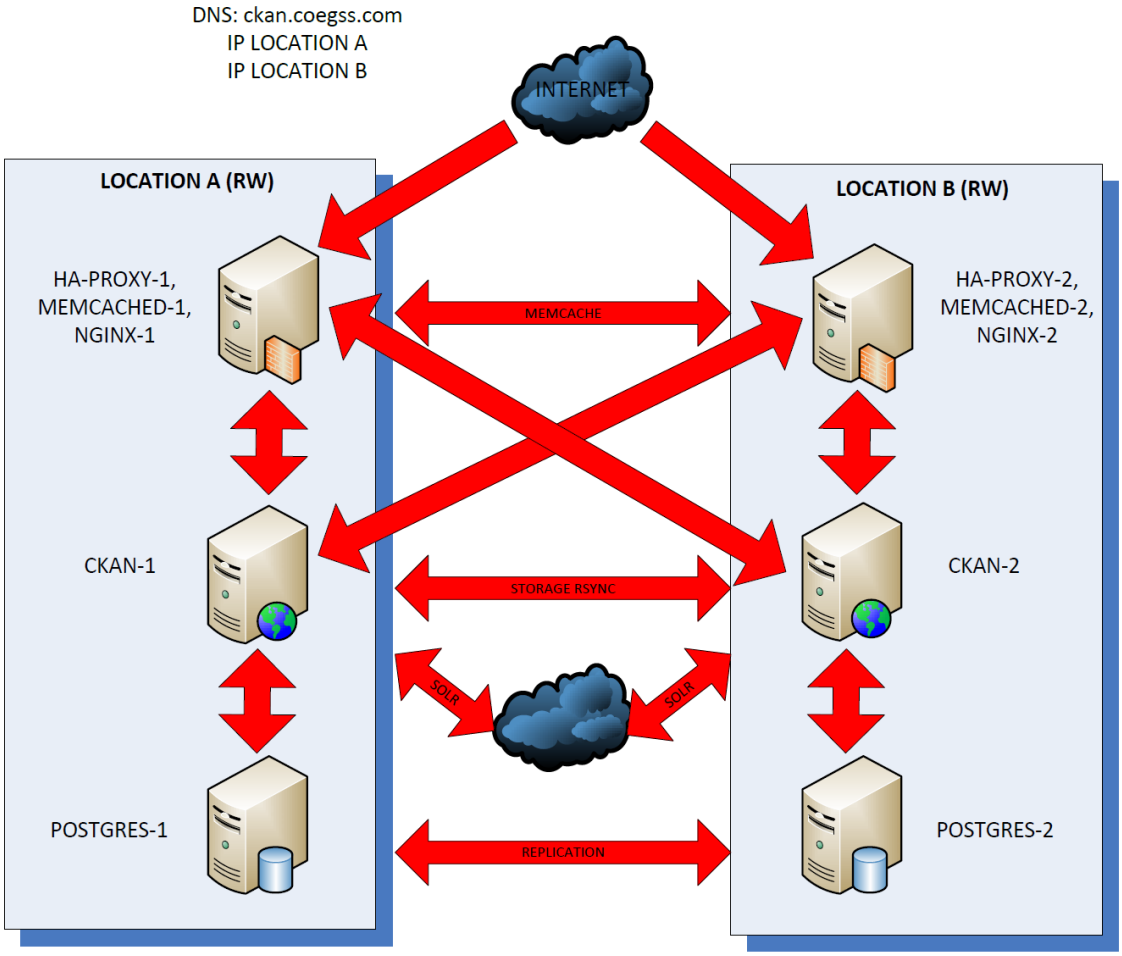

Figure 4: CKAN high availability – locations A and B READ-WRITE

3.3.1.2 Database replication

The CoeGSS project needs to run at least two PostgreSQL database instances – one in location A and one in location B. Recommended solution for database replication is Bi-Directional Replication for PostgreSQL. BDR is multi-master replication system for PostgreSQL specifically designed for use in geographically distributed clusters, using highly efficient asynchronous logical replication, supporting anything from 2 to more than 48 nodes in a distributed database.

3.3.1.3 SOLR replication

The CKAN frontend uses an open source NoSQL search platform from the Apache Lucene project. Its major features include powerful full-text search, hit highlighting, faceted search and analytics, rich document parsing, geospatial search, extensive REST APIs as well as parallel SQL. The Solr can be used as a cloud solution or configured as MASTER-SLAVE instances. In the second case data is replicated only from MASTER to SLAVE node, so the SLAVE instance of CKAN in high availability solution will be READ-ONLY.

3.3.1.4 Storage replication and synchronization

The CKAN FileStore extension saves files to the location set in the CKAN configuration. The storage of CKAN instances can be synchronized through the RSYNC incremental backup mechanism. The synchronization script should run regularly through CRON service. There is possibility to modify CKAN source files to add scripts to execute after a resource is uploaded. This solution should not implicate any security problems.
Alternatively, the CKAN instances can use NFS or any other distributed file system. The NFS is not recommended, as it can be a single point of failure. Distributed file system e.g. CEPH requires many hardware/software and configuration resources relative to a CKAN platform in the CoeGSS project.

3.3.2 Monitoring system

In order to ensure the stability of a system, issues need to be recognised before they become critical. For this purpose, monitoring of services is required, in particular, availability and performance of these services have to be observed. CoeGSS services are being monitored by two monitoring tools, Nagios and Monit. Nagios constantly monitors ping, disk size and service availability for LDAP, HTTP and HTTPS based services for all CoeGSS VMs. All the data gathered from these services and VMs are available for viewing and analysis through Nagios' Graphical User Interface (GUI). Figure 5demonstrates the Nagios GUI with its registered monitoring services, their status, uptimes and status information.

The second monitoring tool deployed for CoeGSS, Monit is being used to recognise service crashes, which helps detecting failing binaries or socket connections. Compared to Nagios, Monit has a different approach, which is simpler but less flexible. For instance, a monitoring probe to check for services can easily be set up by a configuration file, which is interpreted by the Monit daemon. Below, is an example of a configuration file for Monit setting up a monitoring probe for the Apache 2 service which hosts the CoeGSS Portal. The probe checks if the Portal is still alive and if not, attempts to restart the Apache web server and times out after 5 tries.

```
check process apache with pidfile /run/apache2.pid
       start program = "/etc/init.d/apache2 start"
       stop program  = "/etc/init.d/apache2 stop"
       if failed host portal.coegss.hlrs.de port 80
           protocol HTTP request / then restart
       if 5 restarts within 5 cycles then timeout
```

Monit is a lightweight tool, which helps setting up probes quickly that check for failed services and automatically revives them to avoid downtimes. On the other hand, Nagios is a highly parameterisable and an adaptable tool allowing administrators to create automated workflows.

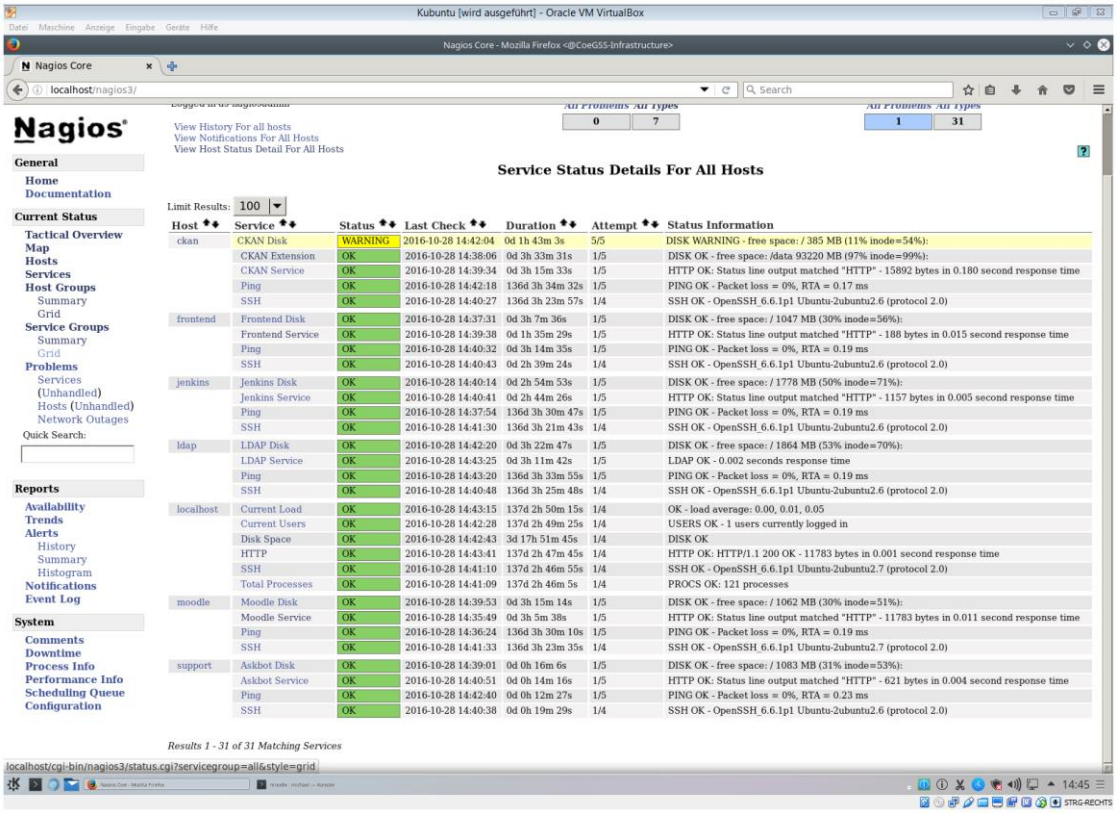

Figure 5: Screenshot of Nagios interface

Using these two monitoring tools at the same time provides a more flexible working environment for the administrators and further simplifies the process of automating systems administration.

3.4 Conclusions

A simulation tool is the heart of many research environments. In the CoeGSS project we have started with the Pandora tool. Efficiency of data processing is crucial to the entire simulation process. Therefore, task 3.1 has spent quite some time on profiling Pandora. Achieved improvements around 24%-70% for 1024 MPI processes allow for much more efficient data processing.

No one has to be convinced how important measurement data are for researchers. For that reason, in the first period of the project work in the task 3.1 was concentrated mainly around providing efficient and reliable data management system. This goal was obtained by introducing the CKAN system in the redundant configuration and the efficient tool (CKAN client) for transferring data between data storage and HPC system. Relevant “crash tests” are still required in order to check how system behaves in failure situations.

Not all system failures can be eliminated by duplicating infrastructure, often administrator reaction is needed. In order to facilitate and boost this process a monitoring system is required. In the CoeGSS project, monitoring is implemented based on the Nagios system. In the upcoming work most crucial CoeGSS services will be covered by Nagios scripts and relevant emergency tests will be performed.

4 Data Management / Data Analytics

4.1 Introduction

This chapter deals with closing the identified gaps related to data management and analysis. These gaps concerned storage and organisation of unstructured data, uncertainty analysis, pre- and processing of data and versioning of data and software.

4.2 Apache Spark

Apache Spark is an open source cluster computing framework, based on the Hadoop ecosystem and originally developed at the University of California, Berkeley's AMPLab.[4] Spark has a strong support base both in academia and in industry. IBM has recently invested in data analytics specific to Spark[5] and Intel is supporting the work on optimising Spark for HPC[6].

Spark offers a fast and parallel-distributed framework for data analytics that can be tightly integrated with an HPC cluster. Another advantage with Spark is that it supplies connectors for different data management systems, including MongoDB, Cassandra, HBase and also for relational databases. Spark also includes MLlib, a machine-learning library for large datasets that provides algorithms such as clustering, regression and Principal Component Analysis. MLlib will be used to perform basic machine learning operations such as clustering the output of the simulations and inferring underlying behaviours in the population.

Currently Spark is used in the Green Growth pilot for analysing the evolution over time of the following three characteristics:

- The number of car types owned by agents of different categories
- The utility of owners of different car types
- The share of agent categories owning a certain car type

This is done by reading the output file from the simulations, transforming it into a specific data structure and then aggregating it. The two lines

```
val df = spark.read.option("inferSchema", "true").option("header", "true")
  .csv("input.csv*")
df.groupBy("carType", "preferenceType", "timeStep").count().rdd
  .saveAsTextFile("output_count")
```

read all files starting with input.csv (e.g. input.csv_1, input.csv_2 etc.), merge them to one data set, distribute this set evenly among the nodes in the cluster used, aggregate the data in parallel so that for each unique combination of car type, preference type and time step the total number of agents is counted and finally collect the aggregated results from the different nodes and write it to disk. The beginning of the output file looks like:

[4] https://amplab.cs.berkeley.edu/software/
[5] http://www.ibm.com/analytics/us/en/technology/spark/
[6] http://insidehpc.com/2015/11/berkeley-lab-to-optimize-spark-for-hpc/.

```
carType,preferenceType,timeStep,count
0,0,0,3613
0,0,1,2412
0,0,2,1735
0,0,3,1241
```

It is worth noting that even though the calculations are done in parallel on a cluster, this is nothing that is seen in the application code, but performed completely automatically.

The results of the analysis are shown in Figure 6.

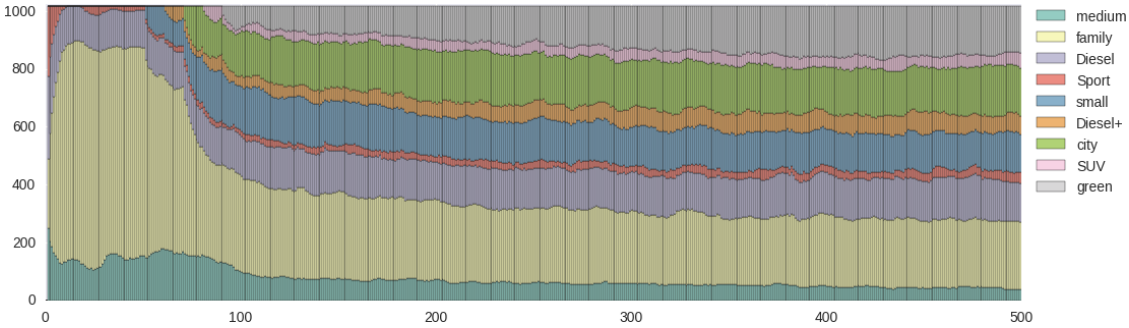

Figure 6: Car fleet stock of the agents for all households over time

4.3 MongoDB

MongoDB is a non-relational, document-oriented database program using JSON-like entries. The database runs on a server and can be replicated or mirrored in slave instances that synchronize for safety.

The program allows for different databases on a single instance and each database may contain different collections (corresponding to the tables of a SQL-like database). Each collection then contains the documents in a JSON-like format, which can be thought of as a Python dictionary or a C hash table (or associative array), i.e., a list of `{key : value}` pairs, binding each key to a corresponding value. While the keys are of string type, the value can be anything from a float, to an array of strings or even another JSON object.

One of the most useful features of MongoDB is that it naturally deals with spatial relationships in terms of GeoJSON-like documents. GeoJSON is a format to encode geographic entities such as points, polygons and lines in a given space (which is usually the `(longitude, latitude)` coordinates system) [7]. In addition, the GeoJSON format allows a geographical feature to be stored within the `'properties'` key the values associated with this region to be later retrieved.

The health habits pilot created two collections in a MongoDB database storing the cells of the Socioeconomic Data and Applications Center (SEDAC) population count raster for 2015[8] and the European countries' boundaries as found in the Nomenclature of Territorial Units for Statistics (NUTS) scheme[9].

[7] See http://geojson.org.
[8] See http://sedac.ciesin.columbia.edu/data/set/gpw-v4-population-count-adjusted-to-2015-unwpp-country-totals.
[9] See http://ec.europa.eu/eurostat/web/gisco for details.

The two collections are organized to provide the following features:

- hierarchical structure of the boundaries to replicate the European Commission's NUTS geographical division;
- store census and national health agencies data for each region;
- fast access to the SEDAC raster cells falling within a region to generate the simulation input rasters for the observable under investigation.
- fast individuation of the boundary containing a specific cell of the raster.

The `cells` collection stores the information about the number of people living in a ∼1x1 Km area around the world as GeoJSON polygons (rectangles in the latitude-longitude coordinates system) whose `'properties'` field reports the population count.

The `boundaries` collection also comprehends polygons delimiting the NUTS at their different levels. The `'_id'` value of these entries is set to the NUTS code which naturally provides a hierarchical organisation of the documents, as the code structure reads CC123, where CC are two characters identifying the country, while 1, 2, 3 are three alphanumeric characters that are present in the first, second and third NUTS level, respectively (i.e., CC12 is a code of level 2 for country CC which is a children of the CC1 level 1 NUTS). The hierarchical organization of the boundaries is obtained by using the NUTS codes from their 0 (lower) to 3 (higher) level as shown in Figure 7.

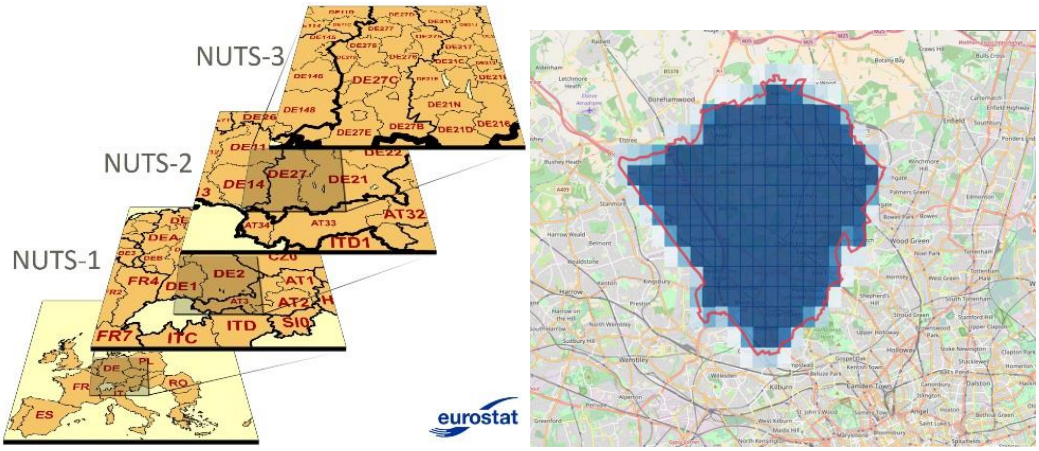

Figure 7: (Left) The NUTS hierarchical levels explained[10]. (Right) The SEDAC cells (blue rectangles, colour proportional to the overlap) intersecting with a LAD in the NW part of London (red edge).

4.4 Database interface

Besides the database, the health habit pilot also developed a high-level interface to insert (retrieve) data to (from) each boundary, aggregate data, import and compare the simulations' output with empirical time series and visualize the results on a map. The interface provides a quick interaction with the data as shown in the following three examples.

[10] Image from http://ec.europa.eu/ eurostat/web/nuts/overview.

Insertion We start from a csv whose columns store the NUTS codes of the regions to which data are referring and the different values we want to include in the dataset for, say, the year 2012. For example, we may have:

NUTScode	CurrentSmokers	ExSmokers	NeverSmokers
UKF11	25%	35%	40%
UKF12	20%	40%	40%
…	…	…	…

We can insert the data using the interface with the csv imported as a data-frame (for example using the `pandas` module in `Python`) and then calling the `insertFromPdDf` method from our library instance `geoClient`:

```
smokingData = pandas.read_csv('smokingPrevalence.csv')
fieldsToInsert = {
         'CurrentSmokers': 'properties.data.health.smoking.2012.CurrSmok',
         'ExSmokers':      'properties.data.health.smoking.2012.ExSmok',
         'NeverSmokers':   'properties.data.health.smoking.2012.NeverSmok'
      }
geoClient.insertFromPdDf(dataFrame=smokingData, keyDF="NUTScode",
          keyDB="_id", fieldsMap=fieldsToInsert)
```

Where `fieldsToInsert` is a dictionary specifying where each column (key) value should be saved in the database record. The latter is given using the *dot-notation*, i.e. `properties.smoking.2012` will be saved in `document['properties']['smoking']['2012']`.

Aggregation Once data are loaded into the database at the NUTS3 level in UK, one may want to aggregate them at the Local Authority District (LAD) or NUTS 2,1, and, 0 levels so as to compute the district, regional and national smoking prevalence, respectively. To do so we perform a weighted average of the smoking prevalence for all the child codes of a given district (for example all the 'UKF1*' are child of 'UKF1') using as weights the population of the child regions, and then repeat the procedure up to the 0-th level. Using our interface this is done by calling the `aggregateCountryLevels` method as:

```
geoClient.aggregateCountryLevels(countryCode="UK", mode="wmean",
    field='properties.data.health.smoking.2012.CurrSmok',
    on="properties.population.total.2015", levelStart=3, levelStop=0)
```

where we specified the `mode='wmean'` weighted-mean aggregation and the on field specifies the document entry to be used as the mean weights. We also instruct the method to start aggregating at level 3 and stop at level 0.

Visualization Data can be easily visualized using a combination of the `bounds2df` method and the `folium` module for map creation. Passing the `pandas` data-frame returned by the former method to the second, we specify the columns to be populated with data stored in the record in a dot notation. For example, to retrieve the smoking prevalence for each LAD in Great Britain in 2012-2015 we can issue the following command:

```
currSmokDF = geoClient.bounds2df({"$and": [{"properties.STAT_LEVL_": 3},
        {"properties.CNTR_CODE": "UK"}]}, useAsIndex="_id",
        record2column={"properties.data.health.smoking.2012.CurrSmok":
            "CurrSmok201*"})
```

Once we have the currSmokDF data-frame we can easily create a choropleth using folium as this:

```
map = folium.Map(location=[54.5,-5.], zoom_start=6)
map.choropleth(geo_str=geoClient.getCountryLevel(country='UK', level=3),
               data=currSmokDF, columns=["index", "CurrSmok2012"])
```

In the latter we set the 'geo_str' resources of the boundaries, from which folium reads the GeoJSON information returned by the getCountryLevel method. We also set the data to be mapped (our data-frame) and the columns to use as data. The result is shown in Figure 8.

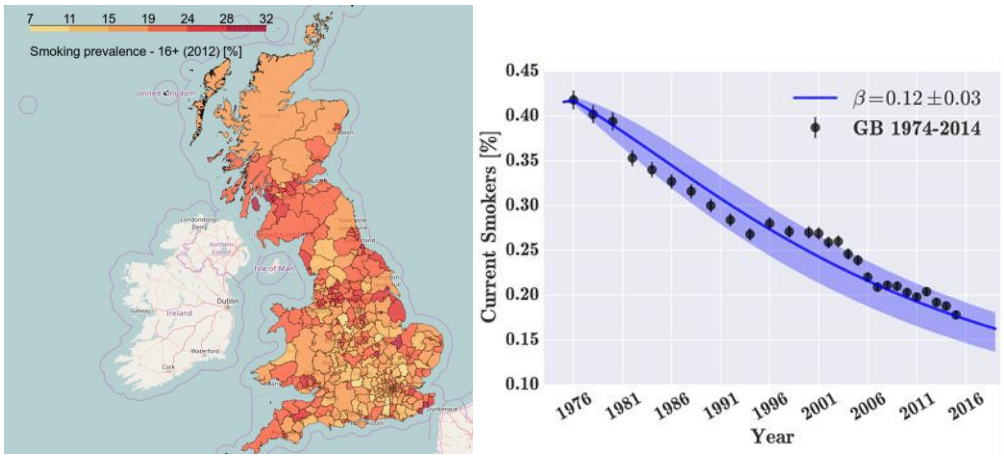

Figure 8: (Left) The smoking prevalence at LAD level resolution in Great Britain in 2012. (Right) The empirical (black dots) and simulated (blue solid line) smoking prevalence in UK from 1974 to 2014. The optimal value of β = 0.12 ± 0.03 is shown together with the confidence interval (blue shaded area).

4.5 Data post-processing

The data post-processing currently consists in the data importing from the output rasters. Specifically, the smoking prevalence in a given region is computed by summing the number of current smokers found in all the cells within the region's boundary, thus reversing the previous boundary to cells mapping of Figure 7 and saving the results in the database using:

```
geoClient.updateBoundary(boundaryID,
  {"$set":
   {'properties.simulations.health.smoking.simulation2012-2020.2012
      .CurrSmok': numberOfCurrentSmokers}})
```

Once this procedure is complete, we can aggregate the results so as to have both the empirical and the simulated smoking prevalence time series, allowing for model calibration.

4.6 Parameter Sweeping

All models contain different numerical parameters that have to be chosen wisely before starting the simulations. Some of these are easy to choose, like the fraction of smokers at any given time, since they are possible to get from available data. Others, like the susceptibility to start to smoke or to buy a green car, given the environment (such as where the agent lives and whom it interacts with) and its socioeconomic properties (such as income and gender) are not always easily available, but have to be calibrated from the model. The most straightforward way to do this is to run the

model for a period in the past, where the expected outcomes are known and then modify the parameters until the simulated process matches the real one. With only one or a few parameters to tune this would be fairly straightforward to do, but as model complexity grows, extensively testing all possible combinations of parameter values quickly becomes unmanageable.

The health habits pilot run a preliminary parameter sweeping for model calibration being that the model features a single free parameter, i.e. , the influence parameter β. To calibrate the model we retrieve the empirical and the simulated national prevalence for the 1974-2014 time interval from the database, for each simulated β value as shown in Figure 8.

Then, for each value of the influence rate parameter β we compute the discrepancy between these two time series as the $\chi^2(\beta)$ sum of squared residuals for each health status compartment, i.e.

$$\chi^2(\beta) = \frac{1}{(N-1)} \sum_{\text{status}} \sum_{y=1974}^{2014} \left[f_{\text{empirical}}(\text{status}, y) - f_{\text{simulation}}(\text{status}, y, \beta) \right]^2,$$

Equation 1

where $f_{\text{empirical}}(\text{status}, \text{year})$ (and $f_{\text{simulation}}(\text{status}, \text{year}, \beta)$) are the empirical (simulated) prevalence of a given health habit *status* at a given *year* (and for a given value of the β parameter for the simulations), respectively. The optimal $\overline{\beta}$ is then defined as:

$$\overline{\beta} = \min_{\beta} \chi^2(\beta).$$

Equation 2

Note that these computations can be done at an arbitrary geographical (NUTS) level (as long as we have empirical data refined at the chosen NUTS level). For example, in the 2012-2015 period data on smoking prevalence are available at the NUTS 3 level (LAD level) in Great Britain. We can then initiate the simulations with the 2012 prevalence resolved at the LAD level, evolve for different values of β and compute the simulated smoking prevalence foe each LAD, separately.

Then, we can back aggregate the smoking prevalence for each LAD and find the optimal beta for each LAD, by generalising Equation 1 to:

$$\begin{aligned} \chi^2_{\text{LAD}}(\beta) &= \frac{1}{(N-1)} \sum_{\text{status}} \sum_{y=2012}^{2015} \left[f_{\text{emp}}(\text{status}, y, \text{LAD}) - f_{\text{sim}}(\text{status}, y, \text{LAD}, \beta) \right]^2, \\ \text{s.t.} \quad \overline{\beta}_{\text{LAD}} &= \min_{\beta} \chi^2_{\text{LAD}}(\beta). \end{aligned}$$

Equation 3

An example of such analysis is reported in Figure 9, where we show the fitted model for the *Dover* LAD and the national map of the $\overline{\beta}_{\text{LAD}}$ in Great Britain.

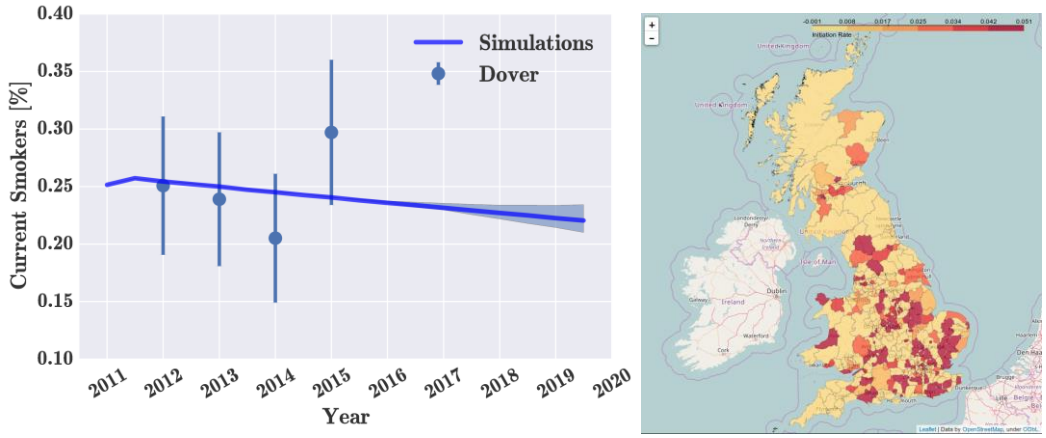

Figure 9: Smoking prevalence analysis. (Left) The empirical (blue dots with confidence interval bars) and simulated (blue line with shaded area) smoking prevalence in the Dover LAD from 2012-2015 (mid 2019 for simulations). The optimal influence rate value is $\bar{\beta}_{\text{Dover}} = 0.05$. (Right) The map of Great Britain with the $\bar{\beta}_{\text{LAD}}$ as obtained from the 2012-2015 model calibration procedure of Equation 3, the redder, the higher the $\bar{\beta}$.

Note that this analysis allows for the localization and highlighting of the national areas where the smoking epidemics is more severe and should then addressed with highest priority.

4.7 Proxy Data and GANs

Traditionally, when data for one region or country is missing, the corresponding data from a similar country has been used as a proxy. What is meant by similar is not obvious, but there is some intuitive understanding of this; for instance, most people would agree that Sweden is more similar to Germany than it is to India. A more systematic approach to deduce similar countries is of course to look at available data and to define similarity in terms of that. If only static data is available but not activity data, which is often the case, principal component analysis or clustering can be used to identify which countries are similar in terms of static properties and then use that knowledge when assigning proxies for activity data. This is of course possible also when other types of data are absent, for instance when only aggregated data is available for a country and proxy micro data has to be used.

Another approach to handling missing data is Generative adversarial networks or GANs. GANs are a type of unsupervised machine learning, where two neural networks are set to compete against each other. While one of the networks is generating synthetic specimens, the other tries to distinguish the synthetic samples from real ones. With training, both networks get better and better at what they are doing and eventually the synthetic samples get hard to distinguish from real ones.

This has successfully been utilized within a number of fields, most notably for generating synthetic images (1), but they have also been used for producing fake health care records (2), in order to provide health records for research without infringing on patient privacy. So far GANs have not been used for generating synthetic populations, but just as fake health care records are motivated

by the wish to be able to work with health records without compromising individual privacy, synthetic populations are, at least partly, motivated by the wish to do agent based simulation on society level without compromising individual privacy. Thus, GANs would be a natural tool to use.

4.8 Conclusions

The different pilots have initially had slightly different focus and made progress in different domains. This has had the advantage that the knowledge base has grown in parallel, but it also introduces the challenge of synchronization and transfer of knowledge. While the green growth pilot has focused on data analytics using Apache Spark, the health habits pilot has focused on the interfacing MongoDB for storing and retrieving geographical data, as well as on calibrating the models using parameter sweeping. The next step will be to systematically transfer the knowledge between the pilots in order to be able to use the acquired within the entire project.

5 Remote and Immersive Visualisation Systems

Within the project, the focus of the Visualization Task is to develop and to provide remote and immersive visualization services to consortium partners as well as to CoeGSS users. These services are continuously being integrated into the CoeGSS portal regarding the defined workflow as described in Section 2.2 (Figure 2). These services provide access to HPC as well as to sophisticated visualization resources integrated in a seamless manner in order to create "Immersive Analytics Environments" for huge statistical and multidimensional datasets.

As a result of the software survey, reported in D3.1, the Collaborative Visualization and Simulation Environment (COVISE) was chosen as a tool to fulfil needed requirements, to handle expected data volume, as well as to be integrated into an HPC environment.

This chapter will list and briefly describe current requirements and development on interfaces to access datasets with respect to the proposed workflow definition as well as modules to read and process datasets for visualization which have been integrated as new services in release 2 of the portal.

5.1 Requirements

Requirements as stated by deliverables and constantly collected from consortium partners as well as users are summarized, refined, prioritized and tracked in a database for further software development. Even though not all requirements can be met within this project, a selection of requirements is given below.

Methods - list of requirements / brief description

Methods	Reference
visualise the results of SI simulations	D4.1 (4.4)
interact with simulation results in real time	D4.1 (4.4)
visualization of full-blown runs, time-series of statistical figures	D4.1 (6.7)
compare multiple runs of the model	D4.1 (6.7)
brush subsets of data points	D4.1 (6.7)
two dimensional maps of cities	D4.1 (7.5)
unfolding different features (population, traffic, prices, pollution, etc.)	D4.1 (7.5)
analysing and interpreting the resulting data (general req.)	D3.1 (4.2)
methods to process huge and varying volumes of unstructured data	D3.1 (4.2)
methods for additional data management	D3.1 (4.2)
Can handle Incomplete information	D3.1 (4.2)
remote visualisation	D3.1 (4.2)
raw mode visualisation	D4.2 (4.1)
visualisation of geo-referenced data on a map	D4.2 (4.2)
compute different aggregations based on shape-files	D4.2 (4.2)
switch between cases	D4.2 (4.2)

Data Interface - list of requirements / brief description

Data Interface	Reference
HDF5 / Pandora Format	
geo data is gridded on a 3432x8640 raster and encoded as geotiff	D4.1 (6.5)
GSS synthetic population simulations	D3.1 (4.2)
Structured and unstructured data	D3.1 (4.2)
Regular and irregular patterns (lists, matrices, graphs)	D3.1 (4.2)
Read CSV	D3.1 (4.2)
I/O modules / general expandability	D3.1 (4.2)
CKAN interface	D1.3 (5.2)
access CKAN data directly	D3.5 (4.3)
access CKAN data by reference	D3.5 (4.3)
automation of defined processing	D3.5 (4.3)
process of data treatment must be tracked	D4.2 (4.1)
automated and generic extraction from a given file	D4.2 (4.2)
support GIS raster data	D4.2 (7.4)
import HDF5 tables	D4.2 (7.4)

Tool Interface - list of requirements / brief description

Tool Interface	Reference
Pandora	
GLEAMviz simulator tool	D4.1 (5.6)
ggobi (http://www.ggobi.org/)	
CoSMo modelling software	
Hadoop	D3.1 (3.3)
Apache Cassandra	D3.1 (3.3)
R Project	
integrated versioning system for data sets	

Data Size: list of requirements / brief description

Data Size	Reference
large populations of up to a hundred millions individuals	D4.1 (5.6)
first tests using a simulation with about 150k agents and 100 time steps	D4.1 (6.7)
support large number of agents (billions) and related data	D4.2 (7.4)

Access: list of requirements / brief description

Access	Reference
access visualization tools on HPC systems	D4.1 (4.3)
web based access to the visualisation	D3.1 (4.2)
data sets can be handled as private	D4.2 (4.1)
hiding parallel MPI code completely from the user	D4.2 (7.1)

5.2 The CoeGSS Visualisation Toolbox

Essential parts of the CoeGSS Visualisation Toolbox are the Collaborative Visualization and Simulation Environment COVISE and the rendering engine OpenCOVER. The rendering engine OpenCOVER provides standard 3D navigation and interaction as expected from 3D visualization tools. Furthermore, OpenCOVER includes a huge variety of plug-ins to handle visualized data interactively. Currently all UI requirements given by users or described by deliverables are being met.

COVISE/OpenCOVER are available open source on GitHub and as binary download for Microsoft Windows, Linux and MacOS X. As part of the CoeGSS Visualisation Toolbox the software is available on the pre- and post-processing servers on HazelHen at HLRS, which are dedicated for data pre- and post-processing including visualization. Also planned is the installation on PSNC's HPC computer system Eagle.

While the software installation including CoeGSS modules on HPC systems is updated in coordination with the users, the GitHub repository is under continuous development. In addition to core functionalities of COVISE and OpenCOVER around 100 plugins and 400 modules are currently available on GitHub. Modules and plugins are usually implemented to integrate user specific demands or external tools and offer new processing or reading capabilities for instance.

COVISE/OpenCOVER modules/plug-ins

COVISE modules	
ReadPandora	read Pandora output files and select parameters/time steps
StackSlices	stack 2D grids generated from time steps along specified axis
DisplaceUsg	displace mesh points in direction and size specified by a parameter
OpenCOVER plug-ins	
PPTAddIn	live export of screen shots into Microsoft PowerPoint/Word

The following paragraphs will give an overview of modules and plugins developed for the CoeGSS Visualisation Toolbox with participation of the pilots.

5.2.1 COVISE Modules

5.2.1.1 ReadPandora

The COVISE module ReadPandora is a reading module based on the COVISE HDF5 reader, which is available already. The HDF5 reader makes use of the libHDF5 library or the HDF5 C++ API description respectively. As reported in Deliverable 4.2, HDF5 is proposed to become the standard format for CoeGSS data. The ReadPandora module currently enables the user to select an HDF5 file and the parameters from within the file. With executing the module, the module reads the data including all time steps or selected time steps respectively into COVISE and offers data output as polygon mesh, which can be read by the renderer directly. The user can select each single time step or start an animation, which cycles through all read time steps.

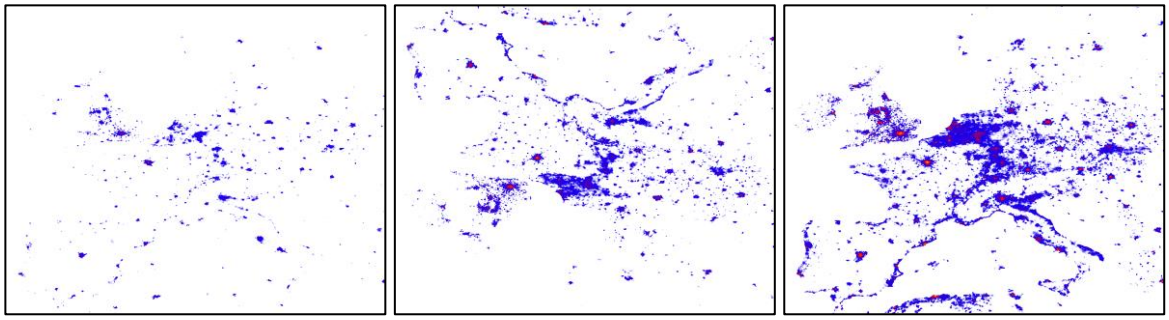

Figure 10: Pilot 2 - Green Growth: Green cars animation loop after 30, 47 & 67 (last) time step.

Storing the data as polygon meshes uses a huge amount of memory but enables quick setup of a visualization and enables further processing, which might be an advantage at this stage of the project. As soon as the CoeGSS workflow has been specified in more detail and the CoeGSS-ABM framework becomes ready, the CoeGSS data format can be specified. Consequently, the data will be read into a more suitable and memory optimized COVISE container then.

5.2.1.2 StackSlices

Usually ReadPandora reads data into data grids, each time step into a separate data grid. In CoeGSS these data grids are usually 2D grids representing geo referenced data on a map. Time steps can be visualised as animation (Figure 10), so the user can see a distribution of parameters over time by mapping colours to the parameter for instance.

Another possible visualisation technique is, instead of using animation over time of a sequence of data sets, to map the time axis or time steps respectively on a selected space axis. The COVISE module StackSlices stacks several uniform grids on top of each other and puts out a 3D data set, which can be visualised using volume rendering for instance (Figure 11). In that case the COVISE Transfer Function Editor (TFE) can be used to map scalar values to a colour schema or transparency.

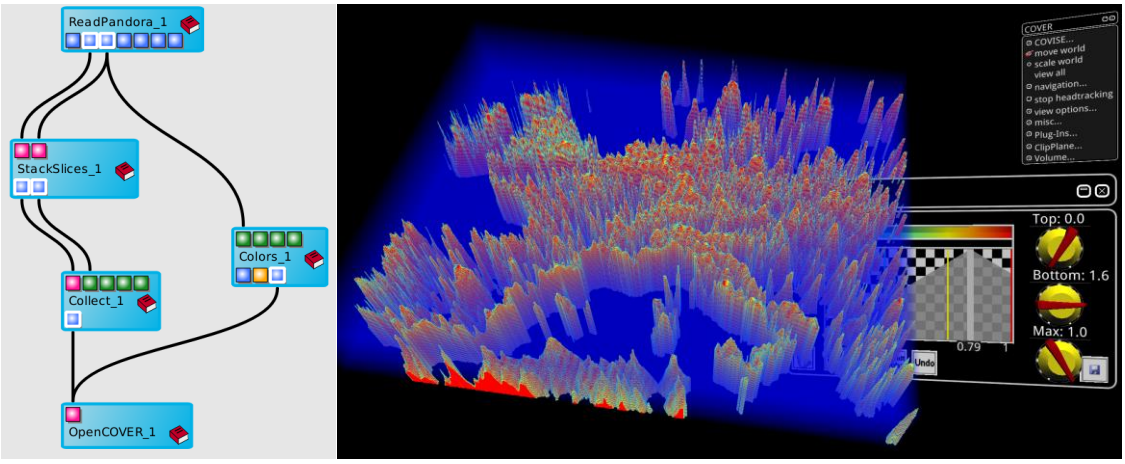

Figure 11: Pilot 2 - Green Growth: Green cars volume rendering

Adding the StackSlices module to the COVISE map-editor the user can define the direction in space to stack the grids as well as the slice distance to create the volume data set. The user now can choose volume handling functionality of COVISE/OpenCOVER for further investigation like clipping planes for instance (Figure 12).

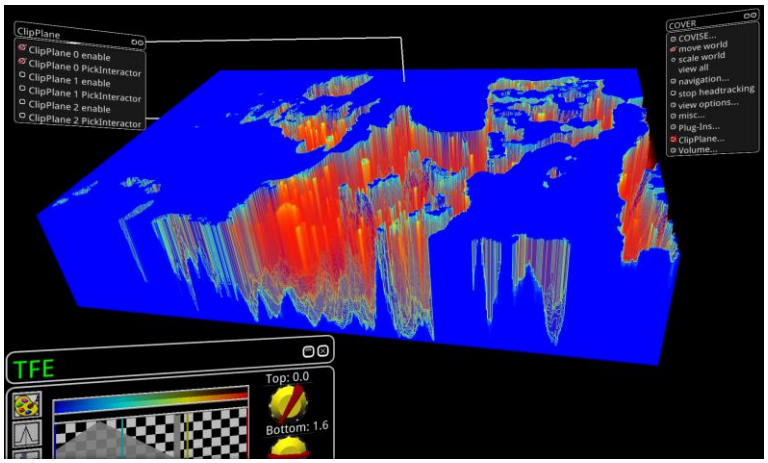

Figure 12: Pilot 2 - Green Growth: Green cars volume rendering with clip plane

A clipping plane clips through the 3D data set and allows a view on the time axis represented by a space axis, so the distribution of properties like the amount of green cars per region over time can be observed.

5.2.1.3 DisplaceUsg

Instead of mapping colours to scalar values within a 2D grid, a height perpendicular to the surface can represent a specific scalar value. So far the COVISE module DisplayUSG used a given vector field for displacement of grid points within the data grid but not scalar values.

Now, the user can choose a specific scalar value from his data set as well as the direction of displacement by choosing a coordinate axis. Furthermore, the user can define a scale operation to map the scalar value to the amount of displacement or height respectively.

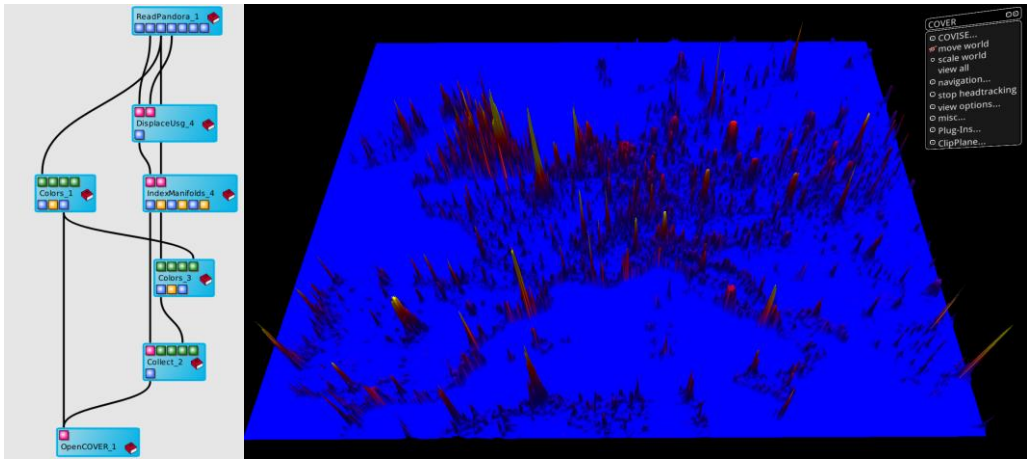

Figure 13: Pilot 2 - Green Growth: Green cars displacement mapping

With using a 2D grid representing geo referenced information on a map, parameters mapped to a colour schema, an animation over time steps and the displacement of the grid points, five dimensions in the data can be observed at the same time.

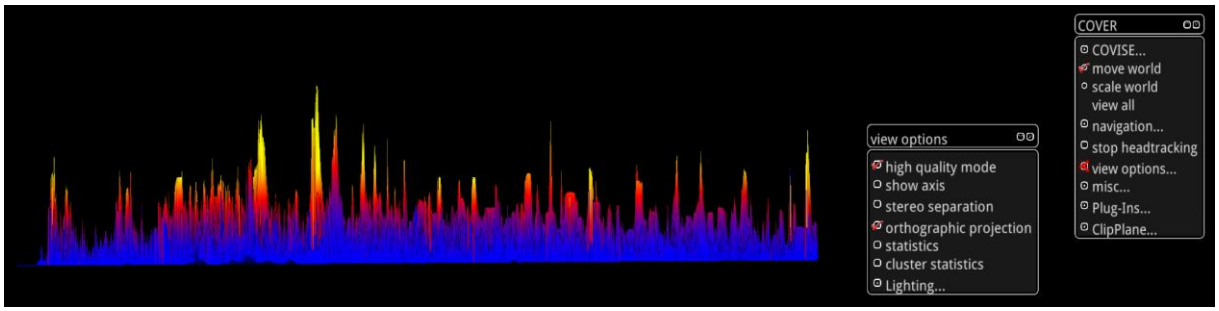

Figure 14: Displacement map in orthographic projection

Along with the possibility to investigate multi-dimensional data sets and dependencies, many other possibilities are being enabled. For instance, with changing to orthographic projection and using clipping planes, this kind of visualisation allows the view on 2D plots through the data set. In an example from Pilot 2 - Green Growth (Figure 14) the height of the graph represents the amount of green cars while the colour mapping is representing the total amount of cars on a projection Western to Eastern Europe. The spikes can be seen as a plot of the amount of green cars in Europe's capital cities.

5.2.2 OpenCOVER Plug-Ins

5.2.2.1 PPTAddIn

The OpenCOVER plugin PPTAddIn was implemented to support the documentation process while using VR environments. While using a CAVE for example, users usually have no laptop or tablet computer at their hands to write down comments or annotations. In fact, the user can add annotation markers to the scene or make screenshots, but these are usually stored on a file system with no context to the VR session. The PPTAddIn plugin enables the user to define and to link screenshots instantly into a Microsoft PowerPoint presentation for example, to track progress of an investigation or data analysis.

Figure 15: Live adding screenshots into Microsoft PowerPoint presentation

5.3 Conclusions

As mentioned this task supports the pilots in developing visualisation tools and capabilities for current data sets, test data sets as well as to setup a visualisation environment, which is being integrated into the CoeGSS Workflow.

Currently COVISE modules and OpenCOVER plug-ins are being further developed and new modules and plug-ins will be developed as necessary to meet user requirements in an agile manner, getting feedback from the pilots constantly. Further on, focus is to test, evaluate and integrate techniques and approaches for the visualisation of huge data sets as well as partial processed data sets.

6 Domain Specific Languages (DSLs)

6.1 Introduction

In this chapter, we report on progress made on closing the gaps identified in Section 6 of deliverable D3.2

In brief, these gaps were:

- lack of reuse of synthetic population data from one pilot to another
- reusing parts of the models implemented using agent-based frameworks
- lack of a common approach to testing

6.2 Data representation

The first gap identified in D3.2, section 6.3, was the lack of reuse of synthetic population data from one pilot to another. The pre-processing phase, which is in general needed in order to use available data from, e.g., Eurostat, is done differently by the different pilots. For example, the Green Growth pilot currently focuses on the global distribution of car ownership, GDP and population density, whereas the Health Habits pilot focuses on smoking prevalence and membership to schools or work places in the UK. The processed data is then not stored in a common format, which would then make it easily accessible to the centre's other applications. Nevertheless, the synthetic individuals have many common attributes. Ideally, we would create a database containing the data needed to create the synthetic individuals, from which each case study could pull the attributes it needs in a uniform fashion. The additional data, if not already present in the database, would be adapted to the CoeGSS format and stored. Our objective is to define this common CoeGSS format and the procedures for regularizing and storing new data.

In order to address this problem, a group dedicated to the task of *Data Management* was formed within the projects, with representatives from the relevant WPs, whose objective is the creation of a common format to store the synthetic population data. The current group's proposal of the common format recommends using the HDF5 file format as the container, and specifies common attributes such as *license*, *sources*, *date* or *version*, which can apply to the whole data set, its individual tables or individual columns. At the same time, the recommendation does not define in detail the formats of the individual fields, as different data sources often use incompatible data formats, which are impossible to normalise without losing information.

During the process of defining a common data representation, further problems have been identified. In addition to descriptions of agents, agent-based models may also make use of relational data, which describes relationships between agents. One example of such a relationship is one agent being a friend of another one. While some relations may be inferred from other data, for example proximity, others require external definition. The relationships may contain attributes, for example we may differentiate between acquaintances and close friends, which leads to a natural representation of them using graphs. Thus, representing large graphs in an efficient way is required.

6.3 Network reconstruction

In general, the available data will not be sufficient to determine the relationships between agents. For example, the Green Growth pilot attempts to model the influence of friends' purchasing behaviour on the agent's own behaviour, but the data that defines "friendship" is largely unavailable (as opposed, say, to the data that defines neighbourhood). In such cases, we face the problem of constructing a graph that is consistent with the available information, while being, in a certain sense, "agnostic" with respect to additional assumptions. We are tackling this problem by means of *network reconstruction methods* via entropy-based models. These are methods that from first principles in Information Theory, provide unbiased way for inferring the missing information. Such methods have been used in dealing with GSS-related fields, e.g., reconstruction of financial network from limited information (3) (4) (5), detecting early signs of world financial crisis (6) (7), or in inferring relationships directly relevant to our agent-based models (e.g., similarity of behaviour resulting from the analysis of Facebook “likes”).

Currently, the IMT group is studying the applicability of these methods in the Green Growth pilot (but they are clearly relevant to the other pilots as well) and released a python package on a GitHub public repository for the implementation of these methods to bipartite networks (an analogous package for monopartite network is going to appear soon). As already mentioned, these theoretical tools are pretty general and can be applied to several different systems. IMT group studied the properties and the effectiveness of such models and is currently investigating their extensibility.

The idea at the basis of elaborated agent based models is that agents sharing several features behave similarly. The IMT group is investigating the possibility of sketching a similarity network among agents from heterogeneous data, like the features defining an agent. The problem lies in comparing data with different entries (binary, integer, real…) and providing a measure that combines them in the proper way. Several similarity measurements for categorical data have been proposed in the literature (8): we are going to review the proposals so far and select those which exhibits highest performances and suit at best the problem at hand.

6.4 Type-based specifications

The second gap identified in D3.2, section 6.3, was the lack of reuse of components of agent-based models.

In the meantime, it has become evident that an approach is needed in not only in order to communicate the structure of the pilot models between the various groups, but also in order to decide on the suitability of software frameworks (such as Pandora), to find and exploit opportunities for parallelisation, and to formulate tests. A description of the model, precise enough yet high-level enough to assist in meeting all these goals, would be a suitable *specification* of the model.

The main idea is to introduce types and functions in order to layer the description of the model. To give a very simple example, the current description of the *Health Habits* pilot states (in the project's internal SVN repository[11], page 2):

> The model can be expressed as a set of differential equation describing the evolution of the system in time:
>
> $$\frac{dS(t)}{dt} = -\beta S(t)\frac{I(t)}{N}$$
> $$\frac{dI(t)}{dt} = +\beta S(t)\frac{I(t)}{N}$$
>
> where the I/N term accounts for the density of infected individuals, i.e., for the probability for a susceptible individual to encounter an I individual.

This is relatively clear, but represents both an over- and an under-specification of the intended model.

First, the functions S and I are introduced here for the first time. From the context, it seems that they are intended to represent the number of individuals that are susceptible and infected, respectively: the number of infected individuals increases with the number of individuals susceptible to infection and with the probability of encountering an infected individual in the population; the number of susceptible individuals decreases correspondingly.

In fact, reading the document it becomes obvious that the last sentence expresses the essence of the model. The differential equations are only one possible *implementation* of the model, and, as it soon becomes apparent, not the best one. In this sense, they are an over-specification: they tell a story that contains more detail than needed, by fixing an explicit mechanism, namely proportional increase. However, they are also an under-specification: we could be modelling both absolute numbers of individuals or fractions in the population and it is not clear which. In either case, the intended model is of functions S and I taking as values natural or rational numbers, but the equations imply that these functions take as values real numbers and are differentiable. The equations provide a (hopefully controlled) distortion of the intended model.

In order to avoid such problems, we propose to start with the introduction of the types of the main functions, and state the most general assumptions made about these. The type of a function f is denoted in the standard way:

```
f : A -> B
```

meaning that the function f associates to every value in its domain A a unique value in the co-domain B. Thus, typing functions implies introducing the sets A and B, i.e., the domain and co-domain. Whereas the expression "type of a function" is standard mathematical usage, the word "type" itself for sets A and B is not. It is, however, standard usage in computer science, and it denotes a set whose elements can be given in a computational way. Since our ultimate goal is the description of computable models, we shall use the word "type" also for the sets involved, even for those which are non-computable (such as the set of real numbers with decidable equality) and, in the course of the implementation, will need to be replaced by computable equivalents.

[11] under `WP04/Documents/healthHabits/17_05_pilotAgentSpecification_v3.pdf`

In the simple example above, we could start with introducing the functions describing the evolution of the subsets of individuals that are susceptible to infection, but not yet infected, and that of the infected individuals:

```
S : Time -> Subset Individual
I : Time -> Subset Individual
```

These functions describe the evolution of subsets of individuals: the evolution of susceptible, but not infected individuals, and that of the infected ones. We have assumed the existence of a time set `Time` and of a set of individuals `Individual`, but we have not made at this level assumptions about whether time is discrete or continuous, etc. This are assumptions that can be made precise at the next layer, but still before the introduction of differential equations.

We can express high-level conditions such as

$$\forall\, t : \mathtt{Time} \;.\; \mathtt{S}\,(t) \cap \mathtt{I}\,(t) = \emptyset$$

(no individual is both non-infected and infected at the same time), or

$$\forall\, t : \mathtt{Time} \;.\; \mathtt{S}\,(t) \cup \mathtt{I}(t) = \mathtt{Individual}$$

(every individual is non-infected or infected). Neither condition is obvious from the description of the model in terms of differential equations.

The description with which we started suggests that neither `S` nor `I` can be computed independently of the other and of parameters such as β. Thus, they are likely to play the role of *data*. For instance, `S t`, the subset of non-infected individuals at time `t`, is the part of the state of the simulation at time `t`.

Many of the data items associated to such a model will be conveniently represented as functions, rather than as, e.g., arrays, lists, or other similar data structures. For example, the age of individuals can be described by a function `age : Individual -> Age`, although in the implementation we might associate to each individual a natural number, and place features such as age or income in arrays, with the age of individual `n` being given by the value of the ages array at index `n`. The functional description abstracts away from the implementation. It has fewer details than the implementation and does not need to change when, for example because of efficiency considerations, the implementation is changed.

Similarly, data about the relationships between individuals, such as “friendship”, will be represented as *relations*, rather than as matrices or lists of lists etc.

A preliminary attempt at using type-based specification for the green growth pilot can be found in the project’s internal SVN repository under `WP04/Documents/GG`, a similar analysis is underway for the health habits pilot, with the urban case study to be tackled next.

6.5 Tools for synthetic population generation

We have continued the investigation of several software packages for synthetic software generation, reported in Deliverable 3.1. Currently, the most promising of these is the open source package *Synthetic Populations and Ecosystems of the World* (SPEW (9)). The software has been considerably improved since our survey, moreover, the developers have shown interest in

collaborating with CoeGSS in the generation of synthetic populations, in particular for the green growth pilot.

Neither SPEW, nor any of the other packages we have examined, is truly "HPC-ready". While SPEW can be run on a cluster of inexpensive machines, the only parallelisation it supports is generating independent synthetic populations in parallel, and thus its performance is limited by the size of the largest synthetic population. As a result, we have decided to re-implement the most important components in order to take advantage of the HPC infrastructures in Stuttgart and in Poznan. The first component to be tackled is the *iterative proportional fitting* (IPF) algorithm, which allows the creation of a synthetic population from a micro-sample and global marginal values.

IPF is a procedure that is used for a contingency matrix to known marginals as one step of synthetic population generation. IPF reconstructs a contingency matrix based on the known marginals in an unbiased way. If lower-dimension contingency matrix is known, their values can be used as marginal for IPF, which then performs fitting that matches the correlations defined by this contingency matrix.

The IPF procedure is often followed by *sampling*, which uses the values from the reconstructed contingency matrix as weights. A synthetic population may be created by sampling from a microsample that comes from the area covered by the marginal data, or from another similar area. IPF may also be followed by the Iterative Proportional Updating (IPU) procedure, which is uses reconstructed contingency matrices for individuals and households, and a micro sample of households containing individuals to perform individual-household assignment.

We have created and benchmarked a proof-of-concept HPC implementation of IPF. The implementation is programmed in C and uses the MPI and PBLAS APIs, which are common HPC APIs whose implementations are available on many platforms. The implementation and benchmarks are described in Deliverable 5.7.

6.6 Conclusions

The work described here is very much "work in progress". At the moment, we do not have a stable version of the database schema for GSS agents, for example. Similarly, the application of network reconstruction to the green growth pilot is yet to be completed, as is the functional specification of the pilots.

An interesting problem with high-level specifications is that a large part of their usefulness is due to being *declarative*, i.e., describing what should be computed, rather than how. This is what facilitates understanding the model, separating the essential from accidental, the interpretation of the results. On the other hand, this also makes it hard to assist the task of parallelisation of the model, which is concerned with the *imperative* aspects: how to best execute the program, taking into account the specific details of the available computational resources. Our approach involves the specification of *acceptable sequences of atomic computations*. The idea is that if two functions `f1, f2 : X -> X` are such that, for a given `x : X` we are indifferent to whether we compute `f1 (f2 x)` or `f2 (f1 x)`, then this is an indication that `f1` and `f2` can be executed concurrently. Note that this indication is obtained without referring explicitly to the implementation, which is different from the standard approaches, such as CPS (concurrent sequential processes), or the Pi calculus.

7 Representing Uncertainty in Modelling and Computation

7.1 Introduction

The main goal of task 3.2 is to ensure the validation and correctness of the software components developed for GSS simulations.

In D3.2 we analysed the requirements of the pilots and identified three gaps in connection with uncertainty representation.

To close these gaps, we proposed to implement an Interval Arithmetic and some interval extensions of optimisation algorithms in a functional programming language. These shall then be used to provide test environments for efficiently running implementations.

7.2 Method

There are several reasons why we decided to represent uncertainty in modelling and computation with the interval arithmetic instead of probabilistic approaches or infinite precision numbers to deal with uncertainty.

Firstly, finite precision numeric calculations are themselves a source of uncertainty. With every step the rounding errors accumulate as loss of accuracy and there is no trace of how big the errors are. With interval arithmetic, however, the loss of accuracy is traced throughout the computation and the exact mathematical outcome of the operation is contained in the result interval. Rounding errors in interval operations are accumulated as the interval width, i.e. as precision loss. Secondly, in many areas of GSS it is quite often the case that the inputs of computations stem from real data and are already grouped into bins or clusters, i.e. we have distributions on intervals like the number of people in the age span 25–30 instead of distributions on single values. Thirdly, we see representation of uncertainty with intervals as a very generic tool. Uncertainty comes from various contexts like structural uncertainty (lack of knowledge), parametric or numerical uncertainty and there are basically two approaches a deterministic one and a probabilistic one. We decided to use a deterministic one (IA) since it can be applied in all cases without making additional assumptions like prior distributions while for example the probabilistic approach can be applied to random variables only and then still has to deal with floating point errors.

The choice to implement the interval arithmetic in a functional language was made because functional languages are very useful to express high level specifications. The implementation of a function in a typed functional programming language is very close to a mathematical function definition and once the properties of it are stated formally, the type checker will provide the proofs of their correctness. Thus they are suited very well to write validated programs.

7.3 State of Work

7.3.1 Interval Arithmetic

The starting point for our implementation of an Interval Arithmetic was IEEE 1788, accepted in 2015[12]. It specifies

> "*basic interval arithmetic (IA) operations selecting and following one of the commonly used mathematical interval models. The standard supports the IEEE 754 floating point formats. Exception conditions are defined, and standard handling of these conditions is specified. Consistency with the interval model is tempered with practical considerations based on input from representatives of vendors, developers and maintainers of existing systems.*"

We started with the implementation of two different data types for intervals in Idris to decide which one will be more suitable later on. The first one is a dependent type, i.e. a family of types with two inputs, the left and the right border (floats including +Inf, -Inf and NaN). An element of this type is basically a proof that the left border is smaller than the right one. For this type we implemented addition and multiplication and proved several properties of it, like monotonicity and commutativity. In the second case we have one single type interval (not a family), but there are total functions on it that allow for every element to derive for example the borders, width and centre. Beside the basic arithmetic operations and operations to evaluate expressions, we also plan the implementation of operations to check semantic properties of intervals like for emptiness and wellformedness.

One problem with validated interval arithmetics is that IEEE 754 allows several rounding modes, which are set "globally". To implement the correct operations, we introduced for every floating point operation two variants as postulates. As an example below is the declaration for the type of the product of two intervals x and x' of the types IntF a b and IntF a' b':

```
multIntF : {a, b, a', b' : Double} ->
           (x  : IntF a  b ) ->
           (x' : IntF a' b') ->
           IntF  (minDList  (oneD x x')  (threeD x x') )
                 (maxDList  (oneU x x')  (threeU x x') )
```

The left border of the product of x and x' is computed by taking the minimum of {a*a', a*b', b'*a', b*b'}, where * is the product of floats in downwards rounding mode (oneD x x' is defined as downwards product a*a', threeD x x' is the list of downwards products a*b', b'*a', b*b'). The right border is the maximum of floating point products in the upwards rounding mode.

For the future we plan to implement a set of operations beside the basic ones that are needed for the implementation of more complex functions. The purpose of these is to have interval extensions of standard functions (like power or polynomials) to implement optimisation algorithms.

The interval extension $F: [\mathbb{R}^n] \to [\mathbb{R}]$ of a real valued function $f: \mathbb{R}^n \to \mathbb{R}$ is characterised by $F([x]) \supseteq \{f(y) | y \in [x]\}$. This characterisation is not a definition, as there may be several functions F (beside the natural interval extension) that fulfil this condition. We are interested in such functions whose outcome intervals are as tight as possible. Especially when an interval occurs

[12] http://standards.ieee.org/findstds/standard/1788-2015.html

several times in a calculation using parameters, and each occurrence is taken independently (dependency problem), the resulting interval may expand unnecessary. To find good functions it will be necessary to use special methods like mean value. The interval extensions of functions will be implemented when needed for special algorithms.

7.3.2 Optimisation algorithms

In the deliverables D4.2 and D4.4 the pilots mention the need for optimisation methods. Optimisation algorithms are needed for the generation of synthetic populations when no real data are available or cannot be used for privacy reasons in the simulations and in future cases with more complex agents to find the optima of the utility functions.

One of the methods that can be used to generate a synthetic population from marginal distributions is the Iterative Proportional Fitting (IPF) algorithm. IPF gets a contingency table and in an iterative process adapts the rows and columns of a matrix to fit the marginal. We plan to implement an interval based variant of it.

The pilots will develop more sophisticated agents for more complex models of global systems. One direction to go is to involve utility functions to determine the decisions of the agents. The aim of the decisions of a rational agent is then to maximise (or otherwise optimise) its utility. To calculate the optimal choices, optimisation algorithms have to be used. Also for finding best fitting parameters in the models it is useful to have such algorithms. Especially in the case of nonlinear optimisation to validate an implementation it is necessary to have test suites, validated pairs of inputs and outputs. These can be created with support of a validated implementation on intervals. It would also help to find appropriate start values, if one would know a suitable interval that definitely contains the solution.

7.3.3 Divide and Conquer

A widely used algorithm scheme that covers a large class of numerical algorithms and especially that of global optimisation is the divide and conquer scheme (D&C).

As preparatory work for the implementation of optimisation algorithms we started with an implementation of the divide and conquer paradigm for several reasons: It is structurally simple but usually not available as a control structure in programming languages. It is computational efficient; many approximation algorithms for NP hard problems with optimal complexity are deduced from a divide and conquer approach. D&C algorithms are naturally adapted for execution in multiprocessor machines since distinct sub-problems can be executed independently, i.e. they are suitable for parallelisation. D&C algorithms are widely used in numerics; for example, the bisection method for finding roots of continuous functions is based on it.

Instead of implementing some special numerical algorithm we started with an algorithm scheme since the crucial points can often be seen clearer from an abstract point of view. For every concrete algorithm that belongs to this scheme it would then only be necessary to prove that each of its components fulfil the axioms to prove its overall correctness.

The D&C program scheme

The divide and conquer paradigm can be expressed as Idris program as follows:

```
data DC : Type -> Type where
 MkDC:  {X : Type} ->
        (Atom    : X -> Type) ->
        (atomDec : (x : X) -> Dec (Atom x)) ->
        (dsolve  : X -> X) ->
        (split   : X -> (X,X)) ->
        (compose : X -> X -> X) ->
        DC X


realise : {X : Type} -> (a : DC X) -> (X -> X)
realise a x with (atomDec x)
       | Yes = dsolve x
       | No  = compose  (realise a (fst (split x)))
                        (realise a (snd (split x)))
```

where *Atom* is a decidable data type with decision function *atomDec*, *dsolve, split* and *compose* are functional expressions. The *realise* is the algorithm that has to be designed. An instance x of a problem is either an *Atom* in which case it can be *solved directly* or more complex. In the latter case we apply a function *split* that splits the problem up into smaller sub-problems, solve the sub-problems (which is expressed by the application of *realise),* and *compose* the solutions of the sub-problems to the solution for the original problem.

To prove that the algorithm scheme is correct we implemented a list of axioms, describing the input and output conditions of the functions *realise, split, dsolve* and *compose*. Correctness here means that given an input x that fulfils the input condition *In,* the output *realise x* fulfils the output condition *Out*. To prove the correctness of any algorithm that is based on this scheme, it is sufficient to show that the implementation of the functions fulfils the axioms.

Smith (10) describes five axioms for the D&C program scheme. Four of them could easily be formulated in Idris. The fifth requirement for the correctness of the algorithm scheme needed more work because of the recursive structure of Idris.

Expressed as an axiom it says that the function *split* decreases the problem size and that this process of decreasing eventually stops. Translated into mathematical terms it means that the subproblem relation is a well-founded partial order on the domain.

To prove the correctness of the scheme it was necessary to implement a new recursion scheme. This has been done by an extension of the module WellFounded. This new recursion scheme has to be used when a concrete split function is implemented.

The next step is to implement a bisection algorithm for finding the root of a function (and thus extrema of its integral) as an instance of the D&C paradigm. The bisection algorithm will be an interval valued function.

7.4 Conclusions

The steps that have been taken so far in Task 3.2 were mostly design decisions for the basic operations of intervals and the minimal set of algebraic properties that would be needed for the implementation of interval versions of optimisation algorithms. They will become visible when the implementation is complete. The next steps that are planned are the following:

- Completion of basic interval arithmetic for both data types.
- Implementation of (tight) interval extensions for some standard functions like power and polynomial with methods like mean value.
- Implementation of an interval version of IPF
- Implementation of Bisection algorithm based on the already implemented D&C scheme.

8 Hardware and software co-design

During the first year of the project, simulation tools for GSS models were constantly in the focus of pilot studies. Agent based modelling has been chosen as a main simulation approach by all three pilots of the Centre. Initial attempts at simulating pilot models on parallel computers have resulted in a thorough specification of requirements to ABM environments for HPC (summarized in D4.2). In this chapter, we analyse existing parallel ABM frameworks through the prism of pilot requirements, identify their gaps, and outline graph-based approach as a potential workaround to fill the gaps in existing ABM solutions. Finally, we present in detail a proof of concept for the graph-based approach to simulation of GSS models.

8.1 Gaps in the existing ABM frameworks for HPC

We have surveyed a number of state-of-the-art ABM frameworks and selected two of them – Pandora and RepastHPC – for further study as the most suitable for use on HPC clusters. Table 3 highlights major features of Pandora and RepastHPC frameworks with respect to coverage of the pilot requirements.

Table 3: Comparative analysis of the ABM frameworks for HPC

	Pandora	RepastHPC
Modelling language	C++	C++
Data exchange mechanism	pure	
Required level of CS expertise	low	high
Agents pool (context)	hash table	hash table
Social relationships modelling	-	graph projector
Data assigned to edges of social graph	-	+
Multiplicity of projectors	-	+
Spatial data representation	hash table of rasters	grid projector with a set of value layers
Dimensionality of environment	2D	any (incl. 2D, 3D)
Distribution of spatial data	static, even	static, even (by default)
Load balancing	-	-
I/O formats	HDF5, raster files (TIFF, JPG, etc.)	NetCDF, CSV
Events scheduling control	hard-coded sequence	dynamic discrete-event scheduler
Post-processing facilities	-	simple aggregations

Even though both frameworks are written in C++ and hide details of low level parallel programming, they require different levels of C++ and parallel programming proficiency from the end users. Pandora provides intuitive API and completely covers parallelization details. In contrast, RepastHPC user must partially take care of data packaging and agents' synchronization to make data consistent in the distributed environment. As a result, it allows to tune data exchange algorithms according to the needs of modeller, but requires additional coding and higher programming competences from the end user.

None of these frameworks completely covers functionality requested in D4.2. In particular, RepastHPC does not have out-of-the-box support of raster images and HDF5 file format. On the other hand, Pandora lacks instruments for modelling social graph relationships.

The common bottleneck for both frameworks is a naive approach to model spatial environment and distribute workload. The frameworks model environment topology by Cartesian grids. In 2D case, environment attributes are represented by dense matrices of the same size. Indices of the matrices correspond to the spatial coordinates and define locations of the grid vertices. During the distributed simulations, Cartesian grid is split evenly between processes. Since amount of computational work in agent based simulation step is proportional to the number of agents, it results in significant load imbalance if agents are distributed very non-uniformly in space. This is a case for CoeGSS pilots where agents usually are highly concentrated in the urban areas and sparsely distributed outside the settlements. Thus, both frameworks violate a requirement to keep workload balanced between processes. The next section shows a way to solve load balancing issue.

8.2 Outline of the graph-based approach to workload distribution

In order to keep workload balanced, the environment grids should be distributed according to the number of agents in its vertices. It can be achieved if we map social graph on the environment grid and partition this mapping with a graph partitioning software. Mapping of the social graph results in a weighted graph, where weight of the vertex equals the number of agents located at the corresponding node of environment grid, and weight of the edge between vertices is proportional to the number of social links between agents assigned to these vertices. In case of short distance communications, this graph can be complemented by edges that represent spatial proximity of vertices (see Figure 16).

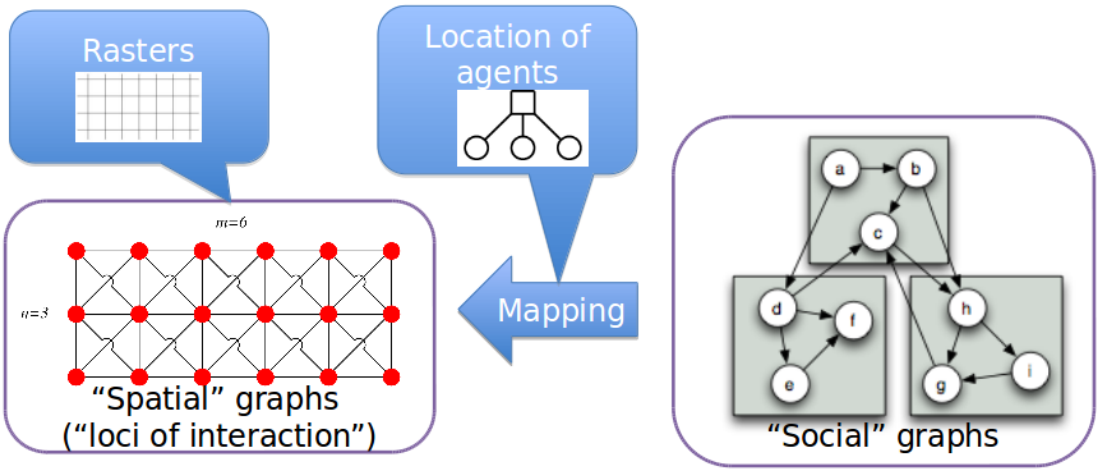

Figure 16: Outline of the graph-based approach

This approach cannot be implemented neither in Pandora nor in RepastHPC without dramatic changes in their cores. In order to implement model according to the graph-based approach without ABM framework, one needs a graph partitioning tool and a general purpose graph library, which provides functionality sufficient to model social relationships.

Table 4 compares potential candidates for a general purpose graph library of choice. PBGL (Parallel Boost Graph Library) is a rather "lightweight" package, which supports most of the features

required to model social graphs (11). Even though interface of PBGL is designed for the users with advanced CS-skills, VTK library provides easy-to-use wraps over native PBGL interfaces (12). PowerGraph is an advanced distributed framework, which implements graph-based gather-apply-scatter (GAS) programming model (13). On the one hand, concept of the graph-based ABM simulation maps perfectly on the GAS programming model. In particular, apply phase allows to specify behaviour of the agents, and gather phase allows to collect suitable information from neighbours. On the other hand, PowerGraph does not support dynamic changes in the graph structure (vertices removal, etc.) (13). The latter strongly limits potential use of PowerGraph for ABM. Both PBGL and PowerGraph assume that all vertices must have the same attributes. Nevertheless, the effect of versatility in vertex attributes can be achieved with variant types.

Table 4: Capabilities of general purpose graph libraries

	PBGL/VTK	PowerGraph
Vertex attributes	+	+
Different types of vertex attributes in the same graph	-	-
Structural information per vertex (degree, adjacencies, node strength, etc.)	+	+
Append vertices/edges	+	+
Remove vertices/edges	+	-
Aggregate attributes	+	+
Iterate over the adjacent vertices	+	+
Iterate over all the nodes	+	+
Group vertices by attribute value	-	-
Adaptive graph repartitioning	+ (user cannot control)	+ (user cannot control)
Input distributed graphs	only POSIX files in format	POSIX files, gzipped files, HDFS files
Output distributed graphs	only POSIX files for	POSIX files, gzipped files, HDFS files

The incomplete list of remarkable graph partitioning packages developed over the last decades includes PT-Scotch, ParMETIS, PaGrid, Chaco, JOSTLE, MiniMax, ParaPART, DRUM, etc. (14). But two of them – METIS and Scotch – gained much more popularity than others and are often referred as load balancing tools of choice in sophisticated time-consuming parallel numerical simulations (14) (15) (16). Table 5 summarizes capabilities of these packages. While both packages fit well to the needs of graph-based approach, ParMETIS is preferable since it allows to repartition distributed graph dynamically.

Table 5: Capabilities of graph partitioning libraries

	ParMETIS	PT-Scotch
Licensing	Own copyleft license	CeCILL-C (LGPL-like)
Partitioning algorithms	multilevel	spectral, combinatorial
Weighted graphs	+	+
Dynamic repartitioning	+	-

8.3 First experiences and results from performance evaluations of distributed graph software

Based on the requirements of the Green Growth pilot as well as the Health Habits pilot to simulate the spread of ideas and social behaviours in complex social networks (social contagion) by agent based modelling, we present in this section the technical background as well as first timing measurements of a very simple proof of concept kernel implementation that realizes a social diffusion process in complex networks based on the agent based modelling approach.

The intention of this work in the context of the current state of the CoeGSS software stack as well as the planned soft- and hardware co-design is, to be able to identify, evaluate and analyse basic but compute intensive operations of which typical algorithms in the area of global system science in general and agent based modelling in particular are composed. In its initial state as it is described in the following sections, the implementation essentially shows the application of several basic operations on complex graphs like iterating over the full node list, operations on all nearest neighbours of a node, graph storage and copying graph data structures.

8.3.1 Basic Idea

The basic idea for the implemented algorithm is inspired by the Green Growth and Health Habits Pilots' modelling approach described in D4.4 and in detail derived from (17) where the value of adoption $\hat{V}$ of a product, behaviour or opinion by a single person k is given by Equation 4:

$$\hat{V}_K = V_k(\hat{\rho}_k) = \frac{\hat{\rho}_k^d}{\hat{\rho}_k^d + \theta^d}(1 + \theta^d)$$

Equation 4

with θ a characteristic constant, d an exponent that determines the steepness of the resulting function and $\hat{\rho}_k$ defined according to Equation 5 as the fraction of users in person k's direct neighbourhood who already adopted the product.

$$\hat{\rho}_k = \frac{n_k}{n}$$

Equation 5

with n the total number of direct neighbours of person k and n_k the number of users who already adopted the product. The value of $\hat{V}_k$ after a number of discrete time steps n_{steps}, as represented in the agent based modelling (ABM) approach, is further on given by:

$$\hat{V}_k(\hat{\rho}_k, n_{steps}) = \max\left(\sum_{i=1}^{n_{steps}} V_k(\hat{\rho}_k, i) \quad , \quad 1 \right)$$

Equation 6

To check the implementation for plausibility on a real-world dataset, we used the Gowalla network[13] provided by the Stanford Network Analysis Project (SNAP)[14]. The network in the used static configuration consists out of $N = 196{,}591$ nodes and $M = 950{,}327$ edges.

8.3.2 Technical details of the initial implementation

For the initial technical realization of the proof of concept implementation we used the C++ libraries SNAP (18) and VTK (12) since they both provide reasonable functionality compared to the requirements of the pilots. Additionally, they both offer serial as well as in the case of SNAP shared memory parallel and in the case of VTK distributed memory parallel graph and data handling algorithms that are planned to be analysed in the future version of this initial implementation.

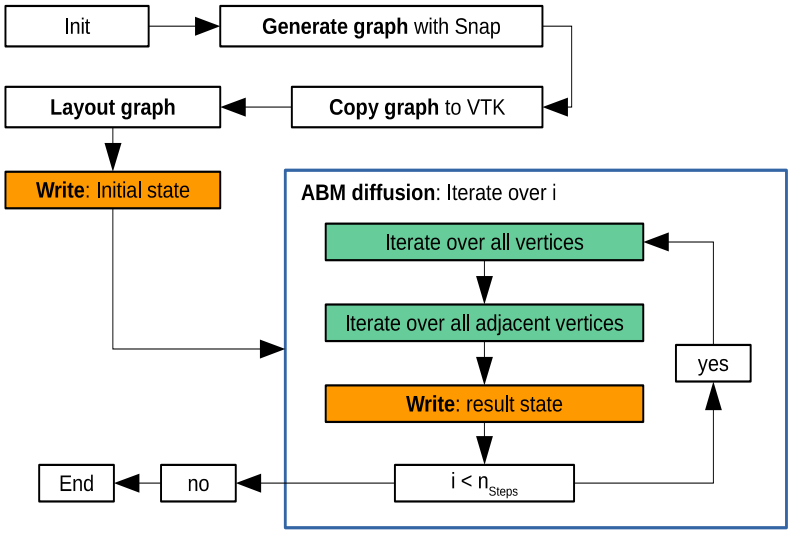

Figure 17: Program flow of the proof of concept implementation

As shown in Figure 17 in the implemented algorithm of social contagion basically 4 significant steps can be identified:

- Generate / Load graph

  Currently three different possibilities to generate or facilitate a given graph by means of SNAP library calls are implemented and tested. These are

  – The generation of a Erdös-Renyi random graph (19),
  – the generation of a Small World graph (20) and
  – reading a given network from file.
- Copy graph

  This step was introduced to see whether it could be reasonable to use two incompatible graph classes for the sake of harvesting algorithms from one library that are not, or only with less efficiency, implemented in the other.

[13] https://snap.stanford.edu/data/loc-gowalla.html
[14] http://snap.stanford.edu

- Write graph

To be able to analyse the results of an ABM model with data-analytics as well as visualization tools, the graph's topology as well as the node properties have to be written to file.

- ABM diffusion

From the perspective of the modeller this step has to be considered the most important one since it introduces the core functionality of an agent based model used for the simulation of social contagion, the algorithm that propagates information through a complex graph. Even though, from a strictly technical perspective, in the current implementation, this step's complexity lacks far behind the one of the other three although this might not directly be visible in the source code[15] due to the used high level advanced programming interfaces (API) of the SNAP and VTK libraries. Technically this step can be implemented in three nested loops.

- Iteration over n_{Steps} time-steps
- Iteration over full node list
- Iteration over adjacent node list

The initial implementation was done in C++ based on advanced programming features like iterators and getter and setter functions as they are favoured by the CoeGSS Pilots' software developers. Also no optimization in terms of extend knowledge about the posed problem is done like e.g. the fact, that the graph's topology is constant over time and by this the inner degree of the nodes does not have to be determined repeatedly in every time-step. Nonetheless, to get an initial idea about the time consuming parts of the code, manual timing measurements of the basic code parts were done by means of the `vtkTimerLog` class as well as with the Cray Performance Analysis Tools (CrayPat).

8.3.3 Timing Results

In this section the first timing results of the initial serial C++ implementation are reported.

Taking a look at Table 6 one recognizes the over 20 times increased time for graph generation in case of the Gowalla network. This has clearly to be accounted to the file access that is done via the high level SNAP API by parsing line by line of the ASCII formatted topology- as well as the check-in-file which contains 6442892 lines of user check-in information. Calculating the effective input bandwidth one finds a value of $55.92\ MB/s$ which shows, that ASCII formatted input / output (I/O) in combination with data selection by parsing is clearly not the way to go for larger file sizes.

Another point that is interesting to see Table 6 is the linear scaling of the ABM diffusion step with the number of nodes in case of the Small World graph type as well as the nonlinear scaling behaviour of the ABM diffusion step in case of the Erdös-Renyi random graph. Additionally, it can be recognized, that the execution time of the ABM diffusion step varies with the regularity of the graph. In case of the Small World graph which has a nearly homogeneous distribution of nodal degrees the ABM diffusion step can be executed the fastest whereas the Erdös-Renyi graph which has the most inhomogeneous distribution of nodal degrees shows the slowest ABM diffusion execution.

[15] http://wiki.coegss.eu/doku.php/....

The Gowalla network lies somewhere in between the other two in execution time of the ABM diffusion step as well as in the variance of the nodal degrees. A potential reason for this correlation as well as for the nonlinear scaling of the ABM diffusion step execution time with the number of nodes in case of the Erdös-Renyi random graph could be the disturbance of prefetching and pipelining mechanisms of the x86 architecture.

Table 6: Manual timing results captured by vtkTimerLog

Graph	Gowalla	Small World	Erdös-Renyi	Small World	Erdös-Renyi
# Nodes	196591	196591	196591	1965910	1965910
# Edges	950327	982955	950327	9829550	9503270
Total Walltime [s]	12.01	3.40	5.07	34.76	59.80
Generate Graph [s]	7.02	0.31	0.25	3.34	4.78
Copy Graph [s]	0.88	0.52	0.62	5.09	7.79
Layout Graph [s]	0.02	0.02	0.02	0.21	0.21
ABM Diffusion [s]	2.38	1.34	3.01	13.64	34.76
Write data [s/step]	0.87	0.60	0.59	6.22	5.67

8.4 Conclusions

The first analysis indicates that existing ABM frameworks for HPC violate some pilot requirements. In particular, one of the common framework bottlenecks is a lack of load balancing algorithms that take into account unevenness in spatial distribution of agents. This chapter presents a graph based approach to ABM simulation with a stack of software that can be used to implement it. Furthermore, with our proof-of-the-concept implementation, we demonstrate that the graph based approach is a useful and efficient mechanism for solving the load balancing issue in distributed ABM frameworks.

One can see that even our initial kernel implementation shows interesting features in terms of GSS application performance on standard x86_64 hardware. E.g., it is obvious that the usage of advanced C++ features not necessarily promotes performance. Also the usage of complex parsing algorithms for loading larger input datasets can significantly reduce I/O performance.

In the next step we plan to introduce the vtkDistributedGraphHelper class to the implementation to get an idea about the parallel scalability of this approach. Further on we plan to re-implement both, the serial as well as the distributed versions of the proof of concept implementation in modern FORTRAN to see at which point the C++ compilers have deficits in generating efficient instruction sets for x86-64- as well as for NEC vector-architecture.

9 Integration of individual components

Within this section of the document, integration of individual components into the workflow, as presented in section 2 are discussed. Consequently, this section defines a clear and sound integration plan by assessing the current state of developments as well as their expected maturity.

In CoeGSS, integration does not only target WP3, but also other work packages. As defined in the Description of Action, WP3 focuses methods and tools for Global Systems Science on High Performance Computing systems. In contrast, WP5 focuses on HPC technology and its performance assessment as well as the development and operation of the CoeGSS Portal. Consequently, integration is defined twofold for CoeGSS: on the one hand, integration of HPC tools and methods directly targets the availability of components on the HPC systems. Whereas on the other, developments of WP3 can also be integrated in the CoeGSS Portal, if it is sensible and requested by the Pilot application stakeholders. All in all, CoeGSS is strictly following the agreed Description of Action, which (implicitly) includes the integration via its defined Milestones (MS) for the technically oriented WP3 and WP5:

- MS3 – Initial Set of Offering identified, M4
- MS4 – First Release of the Portal, M9
- MS5 – Second Release of the Portal with new offerings, M20
- MS7 – Final Release of the Centre as such, M36

The integration concerning components of WP3 is detailed in this deliverable, whereas D1.4 – Second Technical Report will define the entire integration in September 2017. By taking into account the information of the Description of Action, the Milestones, the Deliverables and the specific goals of WP3 and WP5, an integration plan for the WP3 components has been worked out. Therefore, the following Table 7 defines the entire components integration of WP3 with HPC systems but also the CoeGSS Portal.

Component	Type of integration	Deliverable / Milestone	Availability	Description
Tool: Synthetic population generation	HPC	D3.5	Available	The tool for synthetic population generation is primarily integrated on HPC systems. However, control mechanisms for the Portal are planned for M32.
	Portal	D5.13 Portal v4	M32	
Tool: COVISE visualisation	HPC	D3.5	Available	COVISE is a visualisation tool with high performance capabilities. Integration in the Portal can only be handled if the data is processed remotely. So its integration into the Portal is questionable.
Tool: Big Data analytics	HPC	D3.3	Available	Tools for Big Data processing are available. Their integration into the Portal is planned late in the project and still requires some research with respect to data sizes, for example.
	Portal	D5.13 Portal v4	M32	
Tool: Agent based modelling	HPC	D3.4	M24	The agent based modelling and simulation tool is a HPC application. For the moment control mechanisms are foreseen for the Portal as well.
	Portal	D5.13 Portal v4	M32	
Tool: CKAN extensions	HPC	D3.3	Available	The CKAN extensions are available and have been tested on HPC systems already. For the Portal, full integration is planned for M30.
	Portal	D5.12 Portal v3	M30	
Tool: Network reconstruction	HPC	D3.4	M31	Network reconstruction is a complex task, consequently only HPC integration is foreseen.

Method: Workflow integration	HPC	MS7	M34	Full integration of all components within the defined workflow cannot be achieved before the end of the project. Nevertheless, this is not critical since individual components are ready in time and can be used in any case.
Method: Domain Specific Languages	HPC	MS7	M34	DSLs are used to enable the integration of the components. Therefore, full integration is expected in M34.
Method: Interval arithmetic	HPC	D3.4	M31	Interval arithmetic is a research field of big interest for CoeGSS. Its outcomes can be used to define ranges for computation, so that the conceptual models should improve knowledge of the error bars of the Pilot computations.

Table 7: Integration of WP3 components

Summarising Table 7, prototypes of individual components are already available for many tools so that Pilots are able to test the components and provide feedback to the developers. An entire, seamless integration, as detailed above, requires different functionality and involves at least two work packages. The tools' integration into the CoeGSS Portal, for instance, requires particular HPC functionality inside the Portal software stack, so that an initial prototype, which includes a rich set of functionality, can only be expected in M32 of the project's lifetime.

10 Summary

In CoeGSS, WP3 supports the pilots (and, later on, external partners) with research and development of methods, tools and mechanisms (MTMs) for high performance simulations of global systems. In D3.2 (month 6) we presented pilot requirements, gaps and proposed solutions and this deliverable D3.3 (month 21) both describes solutions filling some of the gaps, and specifies new MTMs based on what we have learnt in the project during the last year.

The overall CoeGSS workflow and system architecture was briefly presented in Chapter 2 followed by six chapters on capturing new methods, tools and mechanisms (MTMs) from the point of view of the six tasks of WP3. Chapter 3 presented MTMs for reliability (replication and monitoring), data scalability (CKAN, data storage) and computational scalability. Chapter 4 dealt with data management and data analytics using Apache Spark, MongoDB, database APIs exemplified with pilot use cases for pre- and post-processing of data. Chapter 5 described methods and tools collected in the CoeGSS Visualisation Toolbox using COVISE and OpenCOVER.

With a more theoretical perspective, Chapter 6 described our work on reuse of synthetic population data, network reconstruction, type-based specifications as a common language for reuse of agent-based model components, and tools for synthetic population generation. Chapter 7 focus was on ensuring validity and correctness of CoeGSS' methods and tools. Methods include interval arithmetics, optimisation algorithms, divide and conquer algorithms, and high assurance software through formalisation using types and functions. Chapter 8 presents an approach to agent-based simulation on HPC systems that fills some of the gaps in the existing ABM solutions for HPC. Finally, Chapter 9 addresses the M18 review report recommendations specifically directed at this deliverable.

11 References

1. **Greenemeier, Larry.** When Will Computers Have Common Sense? Ask Facebook. [Online] 2016. https://www.scientificamerican.com/article/when-will-computers-have-common-sense-ask-facebook/#.

2. **Metz, Cade.** Google's Dueling Neural Networks Spar to Get Smarter, No Humans Required. [Online] 2017. https://www.wired.com/2017/04/googles-dueling-neural-networks-spar-get-smarter-no-humans-required/?imm_mid=0f0e59&cmp=em-data-na-na-newsltr_ai_20170417.

3. *Estimating topological properties of weighted networks from limited information.* **Cimini, Giulio, et al.** 4, s.l. : American Physical Society, Oct 2015, Phys. Rev. E, Vol. 92, p. 040802.

4. **Cimini, Giulio, et al.** Reconstructing Topological Properties of Complex Networks Using the Fitness Model. [ed.] Luca Maria Aiello and Daniel McFarland. *Social Informatics: SocInfo 2014 International Workshops, Barcelona, Spain, November 11, 2014, Revised Selected Papers.* Cham : Springer International Publishing, 2015, pp. 323-333.

5. **Mazzarisi, Piero and Lillo, Fabrizio.** Methods for Reconstructing Interbank Networks from Limited Information: A Comparison. [ed.] Frédéric Abergel, et al. *Econophysics and Sociophysics: Recent Progress and Future Directions.* Cham : Springer International Publishing, 2017, pp. 201-215.

6. *Early-warning signals of topological collapse in interbank networks.* **Squartini, T., van Lelyveld, I. and Garlaschelli, D.** #nov# 2013, Scientific Reports, Vol. 3, p. 3357.

7. *Detecting early signs of the 2007--2008 crisis in the world trade.* **Saracco, Fabio, et al.** s.l. : Nature Publishing Group, 2016, Scientific Reports, Vol. 6.

8. *Similarity measures for categorical data: A comparative evaluation.* **Boriah, Shyam, Chandola, Varun and Kumar, Vipin.** In Proceedings of the eighth SIAM International Conference on Data Mining. pp. 243-254.

9. *SPEW: Synthetic Populations and Ecosystems of the World.* **Gallagher, S., et al.** #jan# 2017, ArXiv e-prints.

10. *The Design of Divide and Conquer Algorithms.* **Smith, Douglas R.** Amsterdam, The Netherlands, The Netherlands : Elsevier North-Holland, Inc., #feb# 1985, Sci. Comput. Program., Vol. 5, pp. 37-58. ISSN: 0167-6423.

11. *The Parallel BGL: A Generic Library for Distributed Graph Computations.* **Gregor, Douglas and Lumsdaine, Andrew.** 2005. Parallel Object-Oriented Scientific Computing (POOSC).

12. **Avila, Lisa S., et al.** *The VTK user's guide.* 2010.

13. *PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs.* **Gonzalez, Joseph E., et al.** Hollywood : USENIX, 2012. Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12). pp. 17-30. ISBN: 978-1-931971-96-6.

14. **Schloegel, Kirk, Karypis, George and Kumar, Vipin.** Sourcebook of Parallel Computing. [ed.] Jack Dongarra, et al. *Sourcebook of Parallel Computing.* San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2003, Graph Partitioning for High-performance Scientific Simulations, pp. 491-541.

15. *PT-Scotch: A Tool for Efficient Parallel Graph Ordering.* **Chevalier, C. and Pellegrini, F.** Amsterdam, The Netherlands, The Netherlands : Elsevier Science Publishers B. V., 2008, Parallel Comput., Vol. 34, pp. 318-331. ISSN: 0167-8191.

16. **Karypis, G. and Schloegel, K.** *ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library. Version 4.0.* 2013. p. 32.

17. *Agent-Based Modeling: Methods and Techniques for Simulating Human Systems.* **Bonabeau, Eric.** s.l. : National Academy of Sciences, 2002, Proceedings of the National Academy of Sciences of the United States of America, Vol. 99, pp. 7280-7287. ISSN: 00278424.

18. *SNAP: A General Purpose Network Analysis and Graph Mining Library.* **Leskovec, Jure and Sosic, Rok.** 2016, CoRR, Vol. abs/1606.07550.

19. **Bollobas, B.** *Random Graphs.* [ed.] W. Fulton, et al. s.l. : Cambridge University Press, 2001.

20. *SNAP, Small-world Network Analysis and Partitioning: An open-source parallel graph framework for the exploration of large-scale networks.* **Bader, David A. and Madduri, Kamesh.** s.l. : IEEE, 2008. IPDPS. pp. 1-12.