CoeGSS

Centre of excellence

D3.7 – FINAL REPORT ON METHODS, TOOLS AND MECHANISMS FOR THE FINAL PORTAL RELEASE

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	October 1, 2015
Duration	36 months
Deliverable due date	30.09.2018
Actual date of submission	4.10.2018
Dissemination level	Public
Nature	Report
Version	1.0
Work Package	WP3
Lead beneficiary	PSNC
Responsible scientist/administrator	Marcin Lawenda (PSNC)
Contributor(s)	Piotr Dzierzak (PSNC), Cezar Ionescu, Patrik Jansson, Michał Pałka, Johan Lodin (CHALMERS), Sergiy Gogolenko, Wolfgang Schotte (HLRS), Eva Richter (UP), Fabio Saracco (IMT), Steffen Fuerst (GCF)

Internal reviewers	Sarah Wolf (GCF), Tiziano Squartini (IMT)
Keywords	HPC, Domain Specific Language, Synthetic Information System, Scalability, Visualisation, Co-design, portal, Global Systems Science, HPC, centre of excellence, CoeGSS, data management, synthetic population, synthetic network, data analysis, ABMS, CKAN, Moodle, AskBot, LDAP
Total number of pages:	54

Copyright (c) 2018 Members of the CoeGSS Project.

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/

The information contained in this document represents the views of the CoeGSS as of the date they are published. The CoeGSS does not guarantee that any information contained herein is error-free, or up to date.

THE CoeGSS MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, BY PUBLISHING THIS DOCUMENT.

Version History

	Name	**Partner**	**Date**
From	Marcin Lawenda	PSNC	July 27th, 2018
Contribution from	Michał Pałka, Patrik Janson, Johan Lodin, Fabio Saracco, Wolfgang Schotte, Piotr Dzierżak, Steffen Fuerst, Sergiy Gogolenko, Eva Richter	ATOS, Chalmers, IMT, ISI, HLRS, PSNC, GCF	September 14th, 2018
Version 0.66	for internal review	PSNC	September 21st, 2018
Reviewed by	Sarah Wolf Tiziano Squartini	GCF IMT	September 24th, 2018
Version 1.0 for submission	Marcin Lawenda	PSNC	October 4th, 2018
Approved by	Coordinator	UP	October 4th, 2018

Abstract

This document is foreseen as a continuation of the previous deliverable D3.6 where achievements were addressed in implementing services specified in the deliverable D3.2 as far as they were realized to be integrated in release 3 of the portal. Moreover, it addresses other WP3 achievements in the implementation of methods, tools and mechanisms foreseen by the CoeGSS workflow. The focal point is placed on binding portal and HPC functionality to ensure their correct cooperation.

Table of Contents

Abstract 3
Table of Contents 4
Glossary 5
1 Introduction 8
2 Enhanced Reliability and Scalability 9
3 Data Analytics 14
4 Remote and Immersive Visualisation Systems 18
5 Domain Specific Languages (DSLs) 24
6 Network reconstruction tool 33
7 Representing Uncertainty in Modelling and Computation 37
8 Hardware and software co-design 39
9 Portal – HPC interoperability 45
10 Summary 48
References 49
List of tables 52
List of figures 53

Glossary

3D	Three-dimensional space.
ABM	Agent-Based Model
ABMS	Agent-Based Modelling and Simulation
Apache Spark	an open-source cluster-computing framework
API	Application Programming Interface
AskBot	a popular open source Q&A Internet forum
BLAS	Basic Linear Algebra Subprograms
CAVE	Cave Automatic Virtual Environment (an immersive virtual reality environment)
CFD	Computational Fluid Dynamics
CKAN	Comprehensive Knowledge Archive Network
CMAQ	Community Multiscale Air Quality
CoE	Centre of Excellence
CoeGSS	Centre of Excellence for Global Systems Science
COVISE	Collaborative Visualization and Simulation Environment
CPU	Central Processing Unit
CRB	COVISE Request Broker
CSS	Cascading Style Sheets
CSV	Comma-Separated Values file format
Cuda	a parallel computing platform and API model created by Nvidia
D	Deliverable
DCAT	Data Catalog Vocabulary
DDR (DDR SDRAM)	Double Data Rate Synchronous Dynamic Random-Access Memory
Django	a free and open-source web framework, written in Python
DSL	Domain-Specific Language
EC	European Commission
Eclipse CDT	Eclipse C/C++ Development Tooling
GB	Gigabyte
GB/s	Gigabytes per second
GDDR	Graphics DDR SDRAM
GHz	Gigahertz
GIS	Geographic Information System
GitHub	a web-based VCSs repository and Internet hosting service
GPU	Graphics Processing Unit

GSS	Global Systems Science
HDF5	Hierarchical Data Format, version 5
HPC	High Performance Computing
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IDE	Integrated Development Environment
IP	Internet Protocol
IPF (IPFP)	Iterative Proportional Fitting Procedure
LDAP	Lightweight Directory Access Protocol
M	month
Moodle	a free and open-source software learning management system
MoTMo	Mobility Transition Model
OpenCOVER	Open COVISE Virtual Environment (an integral part of the COVISE visualization and simulation environment)
OpenSWPC	Open-source Seismic Wave Propagation Code
PBS	Portable Batch System (computer software that performs job scheduling)
PDF	Portable Document Format
PostgresSQL (Postgres)	an object-relational database with an emphasis on extensibility and standards compliance
Q&A software (FAQ Service)	a Web service that attempts to answer questions asked by users
RAID	Redundant Array of Independent Disks (a data storage virtualization technology)
RAM	Random-Access Memory
RDF	Resource Description Framework (a family of W3C specifications designed as a metadata model)
SCM	Software Configuration Management
SI	Synthetic Information
SLURM (Slurm)	Simple Linux Utility for Resource Management (workload manager, job scheduler)
SM	Streaming Multiprocessor
SQL	Structured Query Language
SSH	Secure Shell (a cryptographic network protocol)
SSO	Single Sign-On
TCP	Transmission Control Protocol
TB	Terabyte
TOSCA	Topology and Orchestration Specification for Cloud Applications
VCS	Version Control System

VNC	Virtual Network Computing
VR	Virtual Reality
W3C	World Wide Web Consortium
WP	Work Package
XLS	Excel Binary File Format

1 Introduction

With this deliverable, we are providing the knowledge and arrangements we gained and developed since the previous documents, D3.6 & D3.4, which should be taken as complementary when reading this deliverable.

Originally, this deliverable (D3.7) was designed to provide the knowledge from the field of methods, tools and mechanisms designed for the final portal release. As work in other WP3-areas was also continued, we would like to seize this opportunity and report them as well.

This deliverable is organized as follows. Chapter 2 focuses on increasing software realiability by using a dedicated library for checkpointing functionality called Distributed MultiThreaded Checkpointing (DMTCP). Section 3 is on data analytics using the Dakota software. There are two aspects discussed: interoperability with Cloudify and the calibration process. Chapter 4 applies visualization implementation in respect of portal integration. All necessary stages are presented starting from installation and data access through generation of COVISE net-files and optimization to finalizing with running interactive visualization.

In chapter 5, we describe the current status of the domain specific language implementation in the following areas: high-level design, data description and synthetic population generation. The section 5.5 is devoted to describing the synthetic population tool for generating data based on micro-samples. The network reconstruction tool is elaborated in chapter 6. Here we focused on parallelization efforts improving software performance and some aspects towards portal integration. Chapter 7 is about uncertainty efforts and results stemming from the analysis of the n-dimensional IPF algorithm. The co-design achievements are described in section 8. Some information about performance of the HDF5 extension library are provided as well as analysis of benchmarks from the co-design perspective.

In chapter 9, we discuss technical issues on interoperability in Portal-HPC relation with the main focal point on software description using TOSCA (Topology and Orchestration Specification for Cloud Applications) specifications.

2 Enhanced Reliability and Scalability

The calculations and simulations in the CoeGSS project can take long hours. In the event of a software or hardware failure, it is worth to use checkpointing software to not lose the results or start again the application. In this chapter, we report on how we analyzed the available checkpointing software and performed tests.

Software	*Checkpointing method*	*Multithread*	*Programming languages*	*Source code modification*	*Last update*
SWIFT	Application-level checkpointing	NO	C, C++	YES	14.12.2014
Berkeley Lab Checkpoint / Restart	Kernel and software implementation	YES, MPI	C, C++	YES	29.01.2013, v 0.8.5
FTI	Application-level checkpointing	YES	C, Fortran	YES	22.05.2018
DMTCP	**Application-level checkpointing**	**YES, MPI, OpenMP**	**C, C++, Python, Perl**	**NO**	**15.11.2017, v 2.5.2**

Table 1. Checkpointing software overview

We have analyzed the checkpointing software described in Table 1 and selected the DMTCP (Distributed MultiThreaded Checkpointing) tool for testing. It operates directly on the user binary executable, with no Linux kernel modules or other kernel modes. Distributed MultiThreaded Checkpointing transparently checkpoints a single-host or distributed computation in the user-space – with no modifications to user code or to the O/S. Among the Linux applications supported by DMTCP are Open MPI, MATLAB, Python, Perl, and many programming languages and shell scripting languages. DMTCP also supports GNU screen sessions, including vim/cscope and emacs. With the use of TightVNC, it can also checkpoint and restart X-Window applications, as long as they do not use extensions (e.g.: no OpenGL, no video). The DMTCP package enables one to checkpoint a program, and restart it again from the point at which the checkpoint was taken. With DMTCP the following happens when a checkpoint is taken from a running program:

- the program is halted
- the state of the program is written to a file
- the program is started again from the point it was halted

There is the possibility to install the DMTCP software from a Linux repository (Ubuntu, Debian), but we decided to compile and install the latest version from a DMTCP project home page [1].

Firstly, we tested simple programs in C, C++ and Python to see how the DMCTP works. Example of source code of test program in C language:

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    int count = 1;
    while (1) {
        printf(" %2d ", count++);
        fflush(stdout);
        sleep(2);
    }

    return 0;
}
```

Running the program without DMTCP:

```
$./dmtcp1
1   2   3 ^C
```

Running the program with a DMTCP tool:

```
$ dmtcp_launch --interval 5 ./dmtcp1
1   2   3   4   5   6   7   8   9  10 ^C
```

The DMTCP created three new files:

```
$ du -h *
2.7M    ckpt_dmtcp1_76b9252f05c9c5cf-40000-1f1d0c5c5cec3a.dmtcp
12K     dmtcp1
4.0K    dmtcp1.c
16K     dmtcp_restart_script_76b9252f05c9c5cf-40000-1f1d0c1e2a9e7c.sh
4.0K    dmtcp_restart_script.sh
```

The result of the program with a DMTCP restart command:

```
$ dmtcp_restart ckpt_dmtcp1_76b9252f05c9c5cf-40000-1f1d0c5c5cec3a.dmtcp
10  11  12  13  14  15 ^C
```

The program has successfully continued the calculation since the last interruption. The interval parameter specifies how often the checkpoint should be performed. In the restart procedure, we should specify the image file with the extension `.dmtcp`. We performed similar tests for programs written in C++ and Python languages. For all cases, the results were identical.

Finally, we performed tests for matrix multiplication programs written in C++/MPI, C++/OpenMP and Python/MPI. We adopted the matrix size of 5120 rows and columns. The calculations were done on a virtual machine with 48 processor cores (Intel Xeon E312xx Sandy Bridge) and 248 GB of RAM. We made 10 repetitions for each number of threads without DMTCP and with DMTCP with interval 60 seconds and 30 seconds.

	AVG Time [s]								
THREADS	MPI	MPI + DMTCP (interval 60s)	MPI + DMTCP (interval 30s)	OpenMP	OpenMP + DMTCP (interval 60s)	OpenMP + DMTCP (interval 30s)	Python	Python + DMTCP (interval 60s)	Python + DMTCP (interval 30s)
8	799	942	1072	825	936	1078	840	1064	1126
16	439	524	569	423	504	550	443	540	577
32	332	421	464	302	359	373	349	431	469

48	375	525	584	303	362	376	386	551	594

Table 2.2 The average multiplication time for a particular number of threads for C++/MPI, C++/OpenMP and Python/MPI

The best test results of all tests were obtained by C++/OpenMP with 32 threads. In all cases, we obtained the longest calculations on 8 threads, and the shortest on 32 threads. The use of DMTCP software has extended the computation time.

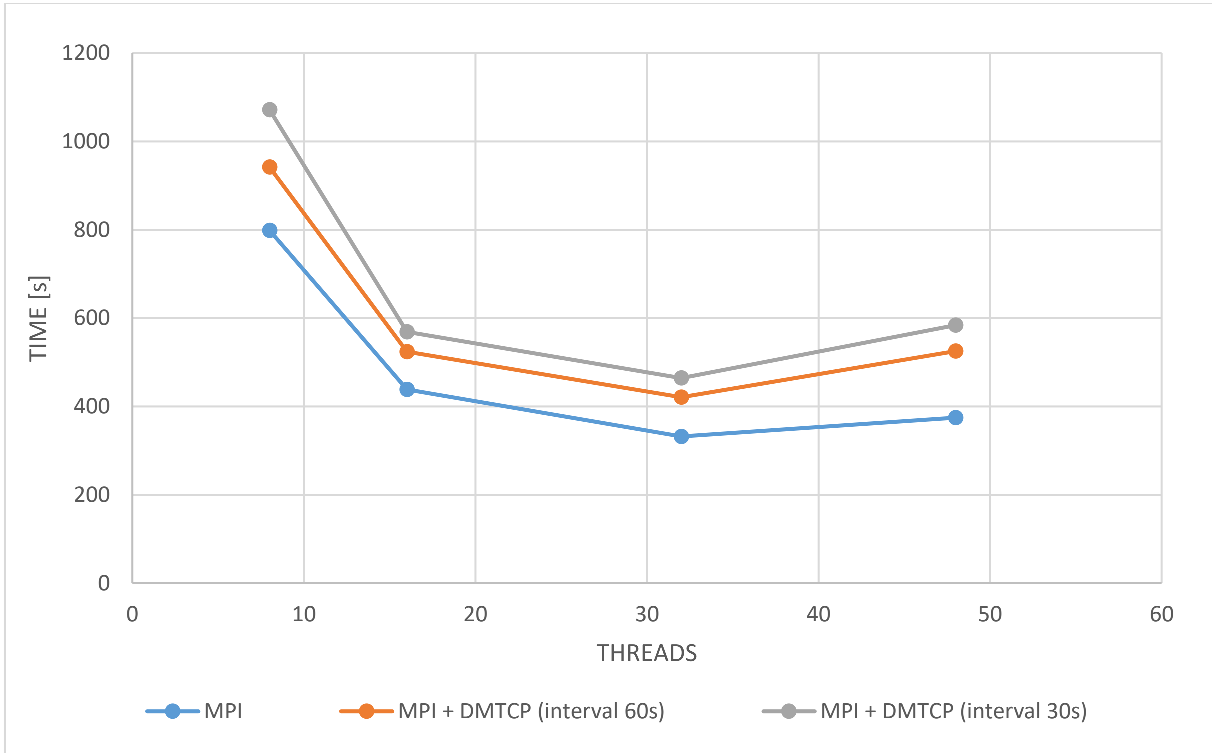

Figure 1. The graph presents the AVG matrix multiplication time for C++/MPI

The point of inflection in the graph is in place of 32 threads. The computation time on 48 threads is slightly larger than the time of 16 with DMTCP tool usage.

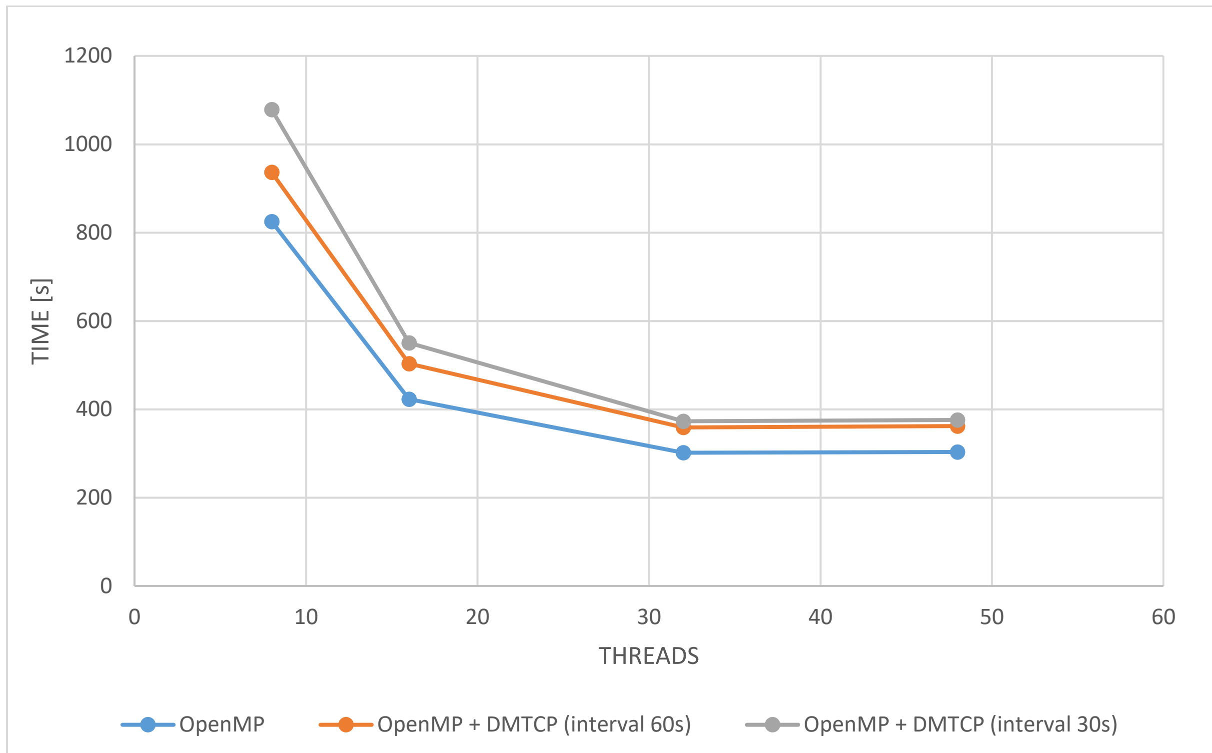

Figure 2. The graph presents the AVG matrix multiplication time for C++/OpenMP

The point of inflection in the graph is in place of 32 threads, but the values of computation time are very similar to the point of 48 threads.

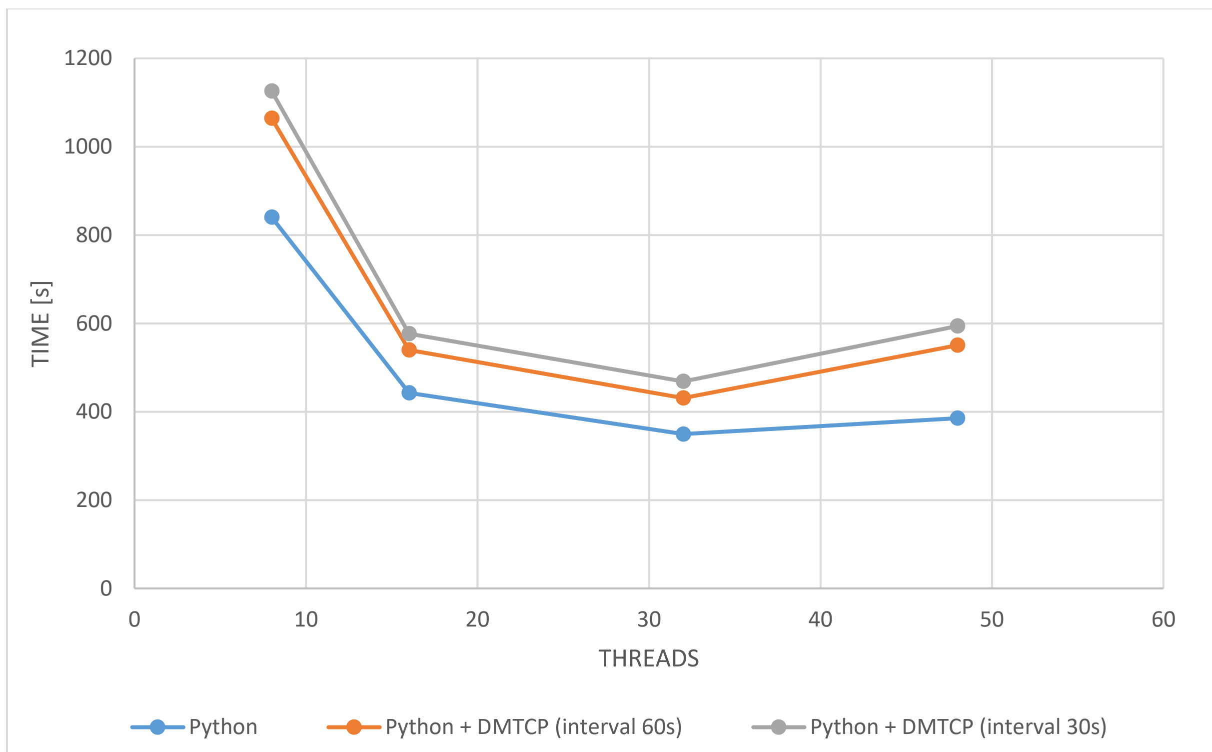

Figure 3. The graph presents the AVG matrix multiplication time for Python/MPI

The point of inflection in the graph is in place of 32 threads. In the case of Python / MPI, the calculation lasted the longest, but at an acceptable level.

Using the DMTCP tool, developers can significantly improve the reliability of the applications. Depending on the number of intervals, the program execution time increases by several or more percent. In our opinion, the value of interval time should be appropriately matched to the size of the task. A heavily extended calculation time can result in more electricity consumption or no completion at the agreed time.

The DMCTP tool can be useful in complex models in the CoeGSS project. It can reduce the time of the calculation and simulation time in the event of a hardware failure.

3 Data Analytics

In Deliverable 3.4, the Dakota [2] software for parameter studies and model calibration is described. In Deliverable 5.12, the Cloudify [3] software for cloud orchestration and a HPC plugin customized for CoeGSS is described. Cloudify provides a way to deploy and execute models on the portal with precise configuration of both the environment and the models, giving an easy way to reproduce both the execution environment as well as the model/Dakota output. In the rest of the chapter we describe how we connect Dakota and Cloudify.

3.1 Dakota – Cloudify interoperability

Cloudify uses so called blueprints, and the Dakota blueprint looks like this:

```
    node_templates:
        first_hpc:
            type: hpc.nodes.Compute
            properties:
                config: { get_input: coegss_hlrs_laki }
                external_monitor_entrypoint: { get_input:
monitor_entrypoint }
                job_prefix: { get_input: job_prefix }
                base_dir: "$HOME"
                workdir_prefix: "dakota"
                skip_cleanup: True
        single_job:
            type: hpc.nodes.job
            properties:
                job_options:
                    type: 'SBATCH'
                    modules:
                        - dakota/6.8
                        - python/2.7.14
                    command: "launch-dakota.sh"
                skip_cleanup: True
            relationships:
                - type: job_contained_in_hpc
                  target: first_hpc
    outputs:
        single_job_name:
            description: single job name in the HPC
            value: { get_attribute: [single_job, job_name] }
```

The blueprint is configured by some inputs, shown by `{ get_input: ... }` in the blueprint. Inputs are not model inputs, but instead blueprint inputs. An example is shown below:

```
    monitor_entrypoint: "193.144.35.146"
    job_prefix: "coegss_"
    # HLRS Laki cluster configuration
    coegss_hlrs_laki:
        credentials:
            host: "cl3fr2.hww.de"
```

```
            user: ""
            password: ""
            login_shell: true
        country_tz: "Europe/Stuttgart"
        workload_manager: "TORQUE"
    # PSNC Eagle cluster configuration
    coegss_psnc_eagle:
        credentials:
            host: "eagle.man.poznan.pl"
            user: ""
            password: ""
            login_shell: false
        country_tz: "Europe/Posnan"
        workload_manager: "SLURM"
```

Here, the user fills in her/his credentials.

When the blueprint is uploaded and deployed, which is done by a sequence of commands detailed in Deliverable 5.12, all files in the directory of the blueprint are also uploaded to the execution server. Among them is the `launch-dakota.sh` script. The specific contents of the script are model dependent, although commonalities exist such as configuration of the workload manager for the target system.

It should be noted that a blueprint is a complete description of the model run, i.e. you do not pass any configuration to the model after the model is deployed. This assures consistency and reproducible results.

Currently, configuration of the model in the Green Growth pilot is done by combining a configuration file written in Python with a Dakota input template. The resulting Dakota input file can then be edited with study specific changes. A portal solution that uses Dakota's GUI would then be limited since the user would work with the generated file rather than the "full" configuration.

As described in D3.6, the model output can be visualized using the R package Shiny [4]. Each model necessarily needs to have its own custom Shiny instance. In combination with the configuration setup above, a GUI would be more user effective compared to using Dakota's GUI directly, by being a model specific configuration GUI (for both the model and the Dakota configuration) that works in tandem with the visualization implemented using Shiny. When a model configuration has been executed, it can then be visualized using Shiny.

3.2 Calibration process

As described in D3.4, section 8.2, Dakota is used for calibrating the agent based SIS "Mobility Transition Model" (MoTMo). Preliminary results are described in D4.6, section 3.2. As

mentioned above, the needed input files for Dakota are generated by a python script[1] which combines templates of those Dakota input files with a configuration description. This is especially useful if you have many parameters or responses which you can generate from a list (e.g. of regions or years). E.g. in the MoTMo case, we fit the car fleet of combustion cars and the car fleet of electric cars for the years 2012-2017 for each federal state of Germany, which results in 192 responses. Writing an input file by hand for those 192 responses would be cumbersome and error-prone.

The work done for integrating Dakota into MoTMo was also carried over to the ABM4py framework, which is described in detail in D3.8. In short, to use Dakota for a model developed with the ABM4py framework, the following steps must be carried out:

- Adjust the model:

The parameters that are set by Dakota must be accessed in the model via `world.getParameter()`. The calls to `world.setParameter()` are ignored when the run is started by Dakota. So, direct assignments of parameters to variables must be rewritten as

```
world.setParameter('imitation', 0.2)
foo = world.getParameter('imitation')
```

- Create a response-script:

The results of a simulation are returned to Dakota in an own python script that is called after the simulation has run. This script must contain one function, `calcResponses`, which gets the response object from the `Dakota.interfacing` python module as a parameter (see section 10.7.4 of the Dakota manual (v6.8) for more details about the response object).

- Create a config file:

This is the small python script mentioned above, which contains `dicts` for e.g. responses, continuous variables, static string …

- Generate the Dakota input file:

Run the script `create-infile-from-template.py` to generate the Dakota input file. Open this file to edit the study specific parts, e.g. the number of samples that should be drawn, or the `evaluation_concurrency`.

- Run Dakota:

This can be done directly by creating a batch script that matches e.g. the `evaluation_concurrency`, or by using the Cloudify blueprint for Dakota, as described in section 3.1.

---

[1] `create-infile-from-template.py` from the `dakota` subfolder of the ABM4py repository: `https://github.com/CoeGSS-Project/abm4py`

More details about the Dakota integration into ABM4py framework are described in the `How-to-use-Dakota.pdf`, which can be found in the `dakota` subfolder of the ABM4py repository.

4 Remote and Immersive Visualisation Systems

Within the project, the focus of the Visualisation Task is to develop and to provide remote and immersive visualisation services to the consortium partners as well as to CoeGSS users. These services are coupled to the CoeGSS portal regarding the defined workflow as described in D3.2 as well as the requirements described in D3.5. A main goal of these services is to provide access to high-performance, highly sophisticated visualisation integrated in a seamless manner in order to create an "Immersive Analytics Environment" for huge statistical and multidimensional datasets.

This subsection will describe the resulting process and final developments on interfaces to access datasets with respect to the proposed workflow definition as well as modules to read, process and visualise given datasets.

4.1 Visualisation Workflow

The visualization tools for remote and immersive visualizations should enable CoeGSS users to analyse large and complex data sets and to make well-founded decisions based on them. The ability to work interactively with the data plays a particularly important role here. This enables a significantly better understanding of the data and enables the user to gain an insight into data relations. Web-based renderers are usually limited in performance as well as functionality and are not suitable for the interactive visualization of such extensive data sets. Furthermore, the IT infrastructure used to host the data usually does not provide the necessary hardware, such as high-performance graphics cards.

For this reason, three different scenarios were proposed by Deliverable D3.5, which describe how the CoeGSS user gets access to data and visualization tools using remote, hybrid or local sessions. Depending on user requirements such as data size for instance, it is possible to view the data on the local laptop, for example, or to use HPC resources, which the user uses anyway to simulate his data. It is also possible to combine both methods with the computing and rendering capacity available on site. The basic operation of the COVISE/OpenCOVER software was described in D3.6 already.

The basic procedure consists of the following steps:

1. installation of the software / gain access authorization to required resources
2. data access
3. downloading the data to the local/remote system
4. generating a COVISE net-file
5. optimizing data processing / preparing the visualisation
6. running an interactive visualisation session

Depending on the kind of session the CoeGSS user wants to initiate, the procedures differ marginally. The procedure is described in more detail below as a general example.

4.1.1 Installation of the software / gain access authorization to required resources

As already described in Deliverable D3.1, the COVISE/OpenCOVER software is available for Windows, Linux and MacOS and can be downloaded from the following website.

https://fs.hlrs.de/projects/covise/support/download/

The source code can be viewed and downloaded from the following website:

https://github.com/hlrs-vis/covise

The documentation of the software COVISE and OpenCOVER as well as tutorials, references and workshop slides are also available online.

https://www.hlrs.de/solutions-services/service-portfolio/visualization/covise/documentation/

If the visualization should run on a local user system, the software has to be installed there. If the CoeGSS user requires using a pre- and post-processing node or a visualization node of an HPC system (see deliverable D3.5), access must be requested in accordance with deliverable D5.1.

After completing this step, the CoeGSS user should have a local COVISE/OpenCOVER installation or should have access to a corresponding remote system.

4.1.2 Data access

Due to the fact that a typical web server infrastructure for an interactive 3D visualization will probably not be able to provide suitable hardware and that the CoeGSS user will get access to HPC resources including any existing visualization frontend anyway, the integration of the rendering frontend into the web server environment seems to be less practical and performant. The interactive remote visualization must assume that the CoeGSS user has the corresponding accesses and resource allocations locally or within an HPC centre.

The CoeGSS user has the possibility to review his data within the portal and can select sections or subsets of his data for visualization. For this data compilation, the CoeGSS user is provided with a data link to his data via which he can access the data from other systems.

With completing this step, the CoeGSS user should have received a link to the data stored or referenced by the CoeGSS portal, which allows the authorized user to download the data.

4.1.3 Downloading the data to the local/remote system

Once the CoeGSS user has received a link to his data, the data must be downloaded from the CoeGSS portal to the local or remote system. Common file transfer protocols such as *wget* or *scp* can be used. This may depend strongly on security restrictions of the local computer or the HPC system.

After completing this step, the data should be accessible by the local or remote system and available for further processing.

4.1.4 Generating a COVISE net-file

If the data is accessible to the machine supposed to run COVISE, the next step is to start COVISE within a graphical environment. Instructions for starting a graphical environment including OpenGL support on the login, visualisation or pre- and postprocessing nodes of a HPC system can be found in the individual descriptions of the system. Depending on the data format used, different reading modules are available. The HDF5 file format, as preferred by the CoeGSS project, can be loaded using the COVISE module *ReadHDF5* or *ReadPandora* (Figure 4).

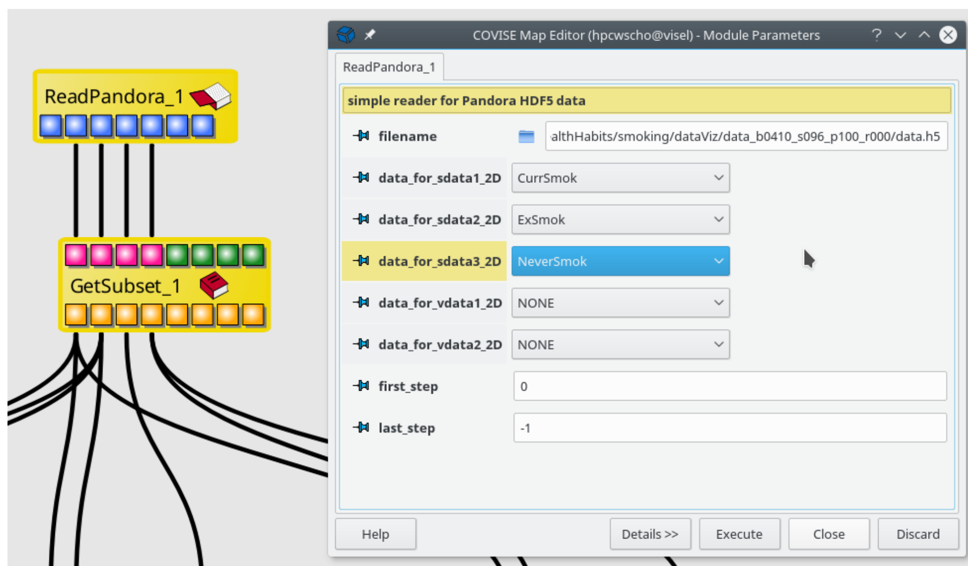

Figure 4. The COVISE ReadPandora module

It is common practice to start with small data sets or a subset to avoid delays caused by initial loading time during development of the first COVISE map. To achieve this, for example, the module *GetSubset* can be used to generate a subset of data (Figure 5).

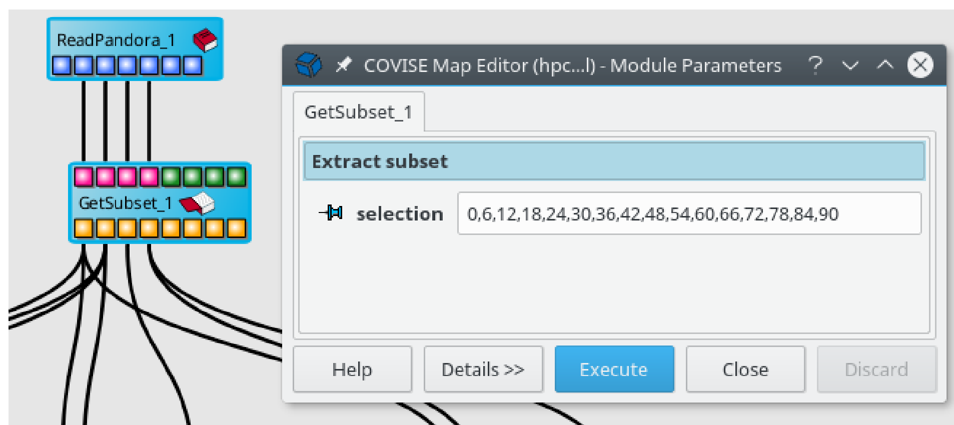

Figure 5. The COVISE GetSubset module

It can be useful to perform a visualization using OpenCOVER in early stages of development by adding the module OpenCOVER to the COVISE map to start the rendering engine.

The goal of this step is to create a first COVISE map that defines the post-processing and prepares the data for the actual interactive visualization.

4.1.5 Optimizing data processing / preparing the visualisation

As already described in deliverable D3.6, various optimization options are available, which are strongly dependent on how the data is structured or has been processed so far, as well as on available computing resources.

A) For example, it is possible to significantly shorten the sometimes very time-consuming reading process in a visualization session if the data selection and reading into a COVISE data container is done in advance. This can be done, for example, by using the GetSubset and RWCovise modules in a separate COVISE map dedicated to run this task (Figure 6).

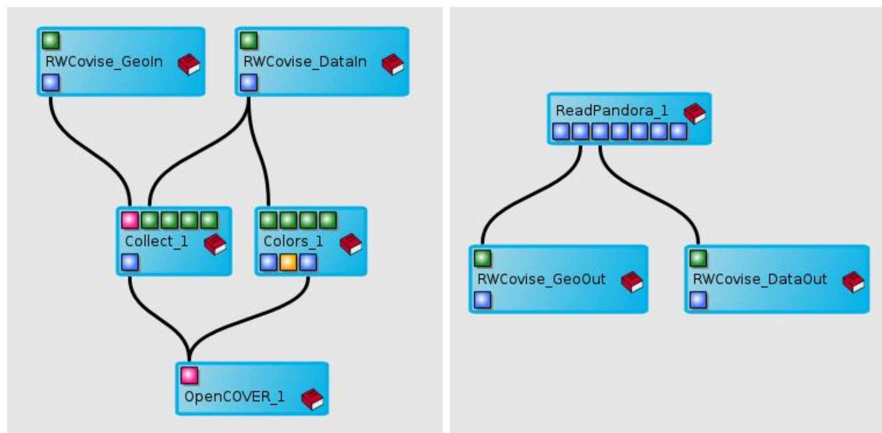

Figure 6. Typical setup of the RWCovise module

Adapting the COVISE map from step 4, now data is no longer read from the raw data (Figure 6, left view), but is read from a COVISE data container (Figure 6, right view). Depending on the data, this can save a considerable amount of time if the raw data is available as ASCII data only, for example, or if it contains a lot of data that is not of interest for the specific visualization.

B) The local session (see deliverable D3.6) can be changed directly into a hybrid session, if, for example, further computers are available for reading and processing the data. Within COVISE, other computers can be added as CSCW hosts, which then take over dedicated tasks.

For example, reading can be done on an I/O node (Figure 7, processes in yellow), processing can be done on a memory node (Figure 7, processes in green) and visualization on a typical laptop (Figure 7, processes in blue) with an adequate graphics card. Certain processes of a COVISE map can be moved on the fly to other computers to which the user has access interconnected with appropriate bandwidth.

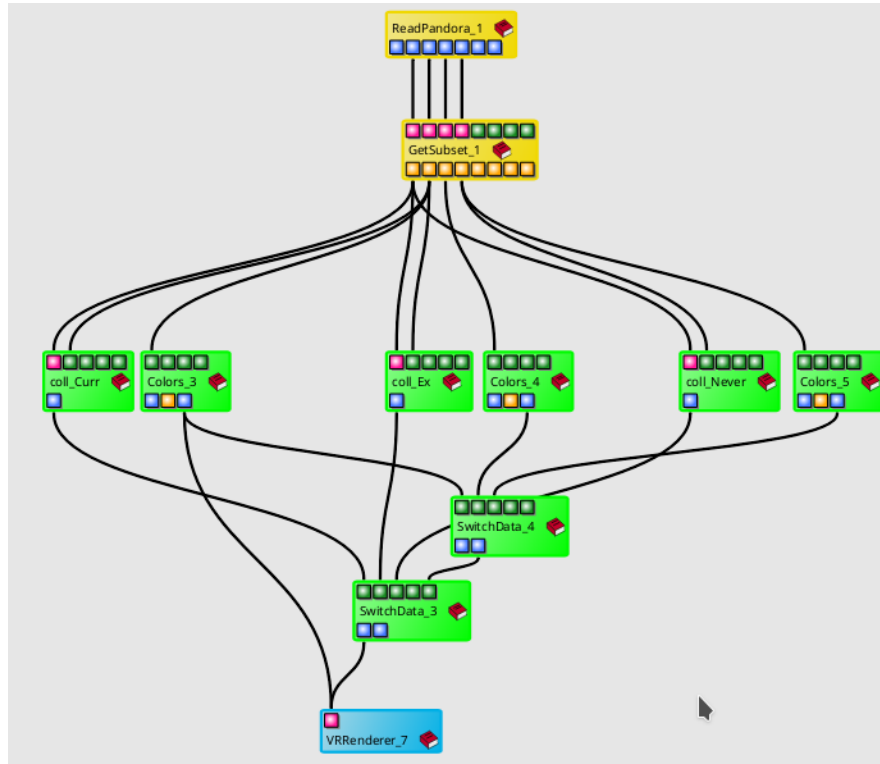

Figure 7. Distributed processing within COVISE

This has several advantages, for example, if the processing of the data requires a machine with more memory or computing power than the local system, the processing can be distributed to high-performance machines within the network.

Finally, with this step, the CoeGSS user should have a running COVISE net-file that can read and process the prepared data and transfer it directly to the visualization.

4.1.6 Running an interactive visualisation session

For visualization of the data, it is finally only necessary to drag the OpenCOVER renderer into the COVISE map editor. This automatically loads and starts the configured visualization frontend. As mentioned in D3.1, it is possible to use OpenCOVER in various visualization environments. For example, it can be configured for an L-Bench, a Powerwall or a CAVE. Examples of such configurations can be found in the COVISE project repository:

https://github.com/hlrs-vis/covise/tree/master/config

In the visualization frontend OpenCOVER, there are various possibilities to interact with the visualisation and the visualised content as well as performing documentation of the user's work within the virtual environment. To control the visualisation, the TableUI (Figure 8) can be used, which represents a 2D-GUI and contains standard control elements as well as control elements for all active plug-ins.

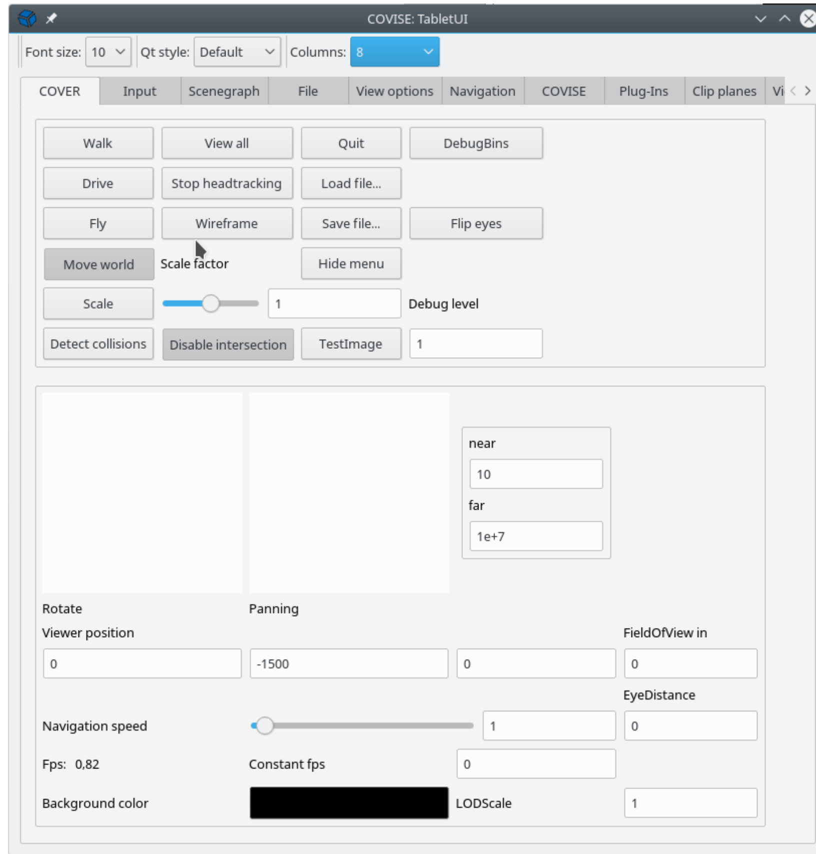

Figure 8. The TableUI standard view

Plug-ins can also be started and used here, for example to take screenshots or video recordings.

4.2 Conclusion

The described workflow enables the CoeGSS user to further examine data uploaded to the CoeGSS portal, review the data as well as process the data and compute simulations by HPC and finally run an interactive visualization session on his data or simulation results. For this purpose, the CoeGSS user can use the software COVISE/OpenCOVER, which focuses on interactive scientific data visualization. The rendering can be realized on different output devices from standard displays over 3D screens up to powerwalls and CAVEs. The software COVISE offers additionally for this project developed or further developed modules and plug-ins, which are adapted to typical GSS data formats or GSS methods. The open source software COVISE and the renderer OpenCOVER offer the possibility to extend and improve these modules as well as the general functionality at any time, so that future visualization requirements can be implemented as required.

5 Domain Specific Languages (DSLs)

Earlier CoeGSS methods deliverables D3.2-D3.3 identified gaps in the functionality provided by state-of-the-art tools that are used for the development of Synthetic Information Systems (SISs), and proposed DSL and network reconstruction tools for addressing the gaps. Deliverable D3.4 presented an evolved design, which was a result of further requirements that had been identified by the pilot projects. Finally, this deliverable describes the implementation status of three of the four DSLs in the evolved design.

We start by describing our DSLs for generating synthetic information systems in the context of GSS simulations. They are:

- **DSLH**: DSL for determining the structure of the simulation needed to answer a high-level question
- **DSLD**: DSL for data description and use
- **DSLP**: DSL for generating the synthetic population
- **DSLA**: DSL for generating an agent-based model. (DSLA is described in the deliverable D3.8, and therefore skipped here to avoid duplication of information.)

These DSLs correspond to different stages of building an SIS for GSS.

An important aspect of GSS models is that the agents are related by multiple networks of relationships: geographic proximity (neighbourhood), but also social and economic connections (friendship, followers on social media, business relationships, etc.). Data about the location of households has been part of census collections since the beginning, and is available from the usual sources, e.g., Eurostat. Social data, on the other hand is much more volatile, and, while the advent of social media has made it more available, its collection is difficult and can lead to skewing results towards certain segments of the population. An alternative is to develop indicators of the influence an agent can have on another's behaviour, and to use the available data to estimate these indicators. Tools to achieve this "network reconstruction" have also been developed within Task 3.4, using the results of **DSLD** and **DSLP** and supporting the construction of ABMs with **DSLA**. These tools are presented in Section 5.5.

5.1 DSLH: DSL for determining the structure of the simulation needed to answer a high-level question

This subsection describes the current implementation of DSLH, the DSL for determining the structure of the simulation needed to answer a high-level question. DSLH is part of the DSLs implemented in Task 3.4 (Domain-specific languages for generating application-specific synthetic populations for GSS simulations). It is presented in the deliverable D3.4 as follows:

DSLH formalises the high-level concepts of "avoidability", "vulnerability", "reachability", associating to each of these the type of simulations needed to assess it.

DSL structure:

- front-end: high-level "avoidability", "vulnerability", "reachability"
- back-end: lists of items needed to assemble the simulation, e.g.
  - the SIS that can be used to determine possible trajectories
  - a harm function, that measures damages, losses, etc. along a trajectory
  - a vulnerability measure that fulfils the monotonicity condition

Further references: *DSLH* is described in the publications “Vulnerability modelling with functional programming and dependent types” [5], “Sequential decision problems, dependent types and generic solutions” [6], “Contributions to a computational theory of policy advice and avoidability” [7].

Current status: The software described in these publications is implemented in the dependently-typed programming language Idris (Brady 2017), and is available at online[2]. A related implementation in the Agda[3] programming language is available at SeqDecProb_Agda[4]. A collection of application independent Idris libraries that grew out of the SeqDecProbs framework is IdrisLibs[5].

The intended use of this DSL is to guide the implementation of lower level software systems. A first example application is outlined in the recent paper “The impact of uncertainty on optimal emission policies” [8].

We have implemented several high-level concepts including reachability, viability and avoidability. Each of these concepts has a computational formalisation in the CoeGSS-Project GitHub repository DSL-Chalmers[6] under the directory SequentialDecisionProblems[7] (see, for example, CoreTheory.lidr[8] for viability and reachability and AvoidabilityTheory.lidr[9] for avoidability).

The structure of simulations needed for computations related to these concepts is given by the unimplemented elements ("*holes*") in these files. They are automatically discovered and highlighted by loading the files in an Idris-aware editor such as Emacs. Examples of

---

[2] SeqDecProbs GitHub https://github.com/nicolabotta/SeqDecProbs
[3] Agda http://wiki.portal.chalmers.se/agda
[4] SeqDecProb_Agda https://github.com/patrikja/SeqDecProb_Agda
[5] IdrisLibs https://gitlab.pik-potsdam.de/botta/IdrisLibs
[6] DSL-Chalmers https://github.com/CoeGSS-Project/DSL-Chalmers
[7] SequentialDecisionProblems https://gitlab.pik-potsdam.de/botta/IdrisLibs/tree/master/SequentialDecisionProblems/
[8] CoreTheory.lidr https://gitlab.pik-potsdam.de/botta/IdrisLibs/tree/master/SequentialDecisionProblems/CoreTheory.lidr
[9] AvoidabilityTheory.lidr https://gitlab.pik-potsdam.de/botta/IdrisLibs/tree/master/SequentialDecisionProblems/AvoidabilityTheory.lidr

implementing viability and reachability are found in the files ViabilityDefaults.lidr[10] and ReachabilityDefaults.lidr[11], respectively. For examples of applications, see the directory SequentialDecisionProblems/applications[12].

5.2 DSLD: DSL for data description and use

The first step towards building an SIS consists in obtaining and processing data. The data describes (some of) the attributes of (some of) the agents that make up the synthetic population. This data is, in general, in "raw" form, containing redundant or irrelevant information. For example, the General Household Survey, 2006 from the UK Office for National Statistics [9] contains a data file with 1541 columns, many of which are not relevant for most applications, and some containing data that is derived from other columns. **DSLD** will allow the users to describe the data required for the agents, describe the raw data, and manipulate the data at a high-level, in terms of agent attributes rather than, e.g., table-columns.

DSLD does *not* manipulate the data directly. Rather, it generates code that can be used to prepare and test the data in the CoeGSS HPC environments.

The DSL has the following structure:

- **Front-end** The user specifies the characteristics and their types (possible values) and relations between them (e.g., if age is < 10 years old, then education level is not University)
- **Back-end** The result is a generated C code for preparing and testing the data

The DSL is architected as a quoted DSL (QDSL) in Haskell, based on the work presented in [10]. Like embedded DSLs (EDSLs), QDSLs allow for borrowing certain aspects of the host language to use in the guest language, such as operator or conditional statements, which allows for fast prototyping.

Unlike EDSLs, QDSL programs are not expressed directly using expressions of the host language, but rather by using quotation, which provides a clear separation between the host language and the DSL. In addition to that, the QDSL infrastructure provides a simple, but powerful optimisation framework, which allows for using high-level constructs in the front-end language, that will then be eliminated during the compilation process, yielding efficient code. For example, all constructions related to sum types, which do not exist in the C programming language, can be eliminated.

---

10 ViabilityDefaults.lidr https://gitlab.pik-potsdam.de/botta/IdrisLibs/tree/master/SequentialDecisionProblems/ViabilityDefaults.lidr

11 ReachabilityDefaults.lidr https://gitlab.pik-potsdam.de/botta/IdrisLibs/tree/master/SequentialDecisionProblems/ReachabilityDefaults.lidr

12 SequentialDecisionProblems/applications https://gitlab.pik-potsdam.de/botta/IdrisLibs/tree/master/SequentialDecisionProblems/applications

To demonstrate how high-level features can be eliminated from the target code, let's consider this definition of a mapping of numeric values contained in a column of a data file into descriptive labels.

```
eduMapQ :: Mapping
eduMapQ =
  [[6, 8, 9]     |-> "NoData",
   [1]           |-> "HigherEd",
   [2]           |-> "OtherEd",
   [3]           |-> "NoEd"]
  `noDefCase` ()
```

The column describes the education level of an individual. If a row contains number 6, 8 or 9 in the column, this means that no data on the education level was collected. Values 1, 2 and 3 denote different education levels. Finally, any other value is illegal, which is specified using the `noDefCase` combinatory.

Having defined the mapping, we can define a property that will operate on the labels given to the values in the column.

```
prop1 :: Maybe String  -> Bool
prop1 (Just "NoData")   = False
prop1 (Just "HigherEd") = True
prop1 (Just "OtherEd")  = True
prop1 (Just "NoEd")     = True
prop1 Nothing           = False
```

The property checks whether there are no illegal values in the column by returning `False` for the argument `Nothing`, which denotes an illegal value. Furthermore, the property also checks whether there are no persons for which the data is missing, by returning `False` for the argument `Just "NoData"`.

Using our framework, the mapping and the property can be combined and compiled down to an efficient low-level C function, as explained in D3.4.

Thus, it is possible to use high-level code on the programmer's side, and at the same time generate efficient low-level code suitable for high-performance compilation.

Currently, a proof-of-concept implementation is available[13], that allows defining basic properties, and generating efficient C code for them. The project consists of Haskell code, in addition to embedded skeleton C code. The implementation uses a custom fork QHaskell[14] library as the optimisation backend, and the language-c-quote[15] library for generating C code.

[13] DSL-tests https://github.com/michalpalka/DSL-tests
[14] QHaskell https://github.com/solrun/QHaskell
[15] Objective-C quasiquoting library http://hackage.haskell.org/package/language-c-quote

5.3 DSLP: DSL for generating the synthetic population

The data available will define, in most cases, only a small number of the agents necessary for the SIS. The next step is therefore to build a synthetic population from which all agents can be defined. This step requires the results of **DSLD**, namely the data description in terms of high-level agent attributes, together with a micro-sample conforming to this description, and statistical information about the population, usually in the form of marginal distributions of attribute values. **DSLP** will then generate *HPC-ready* code for extending the micro-sample to a synthetic population.

The DSL has the following structure:

- **Front-end** The user specifies the input data description using DSLD, and the procedure for generating the synthetic population using a set of primitive operations, like IPF or sampling
- **Back-end** The result is a generation procedure implemented in C

Currently, the following computational kernels are available:

- Efficient IPF kernel based on the PBLAS [9] API
- Haskell implementation created for validation purposes

Just as **DSLD**, also **DSLP** is implemented as a Haskell-based quoted DSL.

Below is an example program written in the DSL, which generates a basic synthetic population.

```
edu_table = marginalTable "edu_table.csv" ["isced97"]
pop_table = marginalTable "pop_table.csv" ["age", "sex"]

marginals = [edu_table, pop_table]
microSample = microSampleTable "microosample.csv"
  ["Sex", "age", "EDLEV10", "GREARN"]

mappingAge = ranges
  [[0, 15]   |-> "Y_LT15",
   [16, 29]  |-> "Y16-29",
   [30, 59]  |-> "Y30-59",
   [60, 74]  |-> "Y60-79",
   [75, 120] |-> "Y_GE75"
  ]

mainSynPop :: IO ()
mainSynPop =
  forM_ ["UKC11 ", "UKC12 ", "UKC13 ",
         "UKC14 ", "UKC21 "] $ \region ->
    generatePop marginals microSample
      (mappings [("sex", "Sex", Id),
                 ("age", "age", mappingAge),
```

```
                    ("isced97", "EDLEV10", Id)])
          ("out_" ++ removeSpace region ++ ".csv")
```

The program specifies the file names or identifiers of the data files containing the marginal distributions, and the data file containing the micro sample. In addition to that, the program needs to specify the relevant column names (as in general the data files might contain more columns), mappings between column names in the marginals data files and the ones from the micro-sample, and mappings of between values (and possibly ranges) from different files.

Finally, the program also specifies that the generation procedure will be performed for 5 regions (UKC11, UKC12, etc.), and will create a separate output file for each of them. Based on this description, the generation algorithm will first perform binning of the micro sample into the cross product of the marginals tables. Secondly, it will use the IPF method to fit the resulting matrix of counts to the marginals according to maximum likelihood estimation. Finally, it will perform sampling from the micro sample based on the estimated weights.

Even though the DSL allows for writing executable programs, many of the concepts appearing in them, such as marginals, micro samples, mappings, etc., are familiar to domain experts, who are not necessarily programmers.

Such compiled program will perform separate invocations of its computational kernel, which uses the Python Numpy[16] library and BLAS[17] computational library under the hood, for every region.

As this DSL is built on top of DSLD, it shares much the same framework as the other DSL.

[16] NumPy http://www.numpy.org
[17] BLAS http://www.netlib.org/blas/

5.3.1 Integration with the Portal

Integration with the Portal is realised through Cloudify, which allows for submitting computational jobs on clusters and HPC machines, as described in Deliverable 5.12. The process of running an application involves 5 components, as shown in Figure 9.

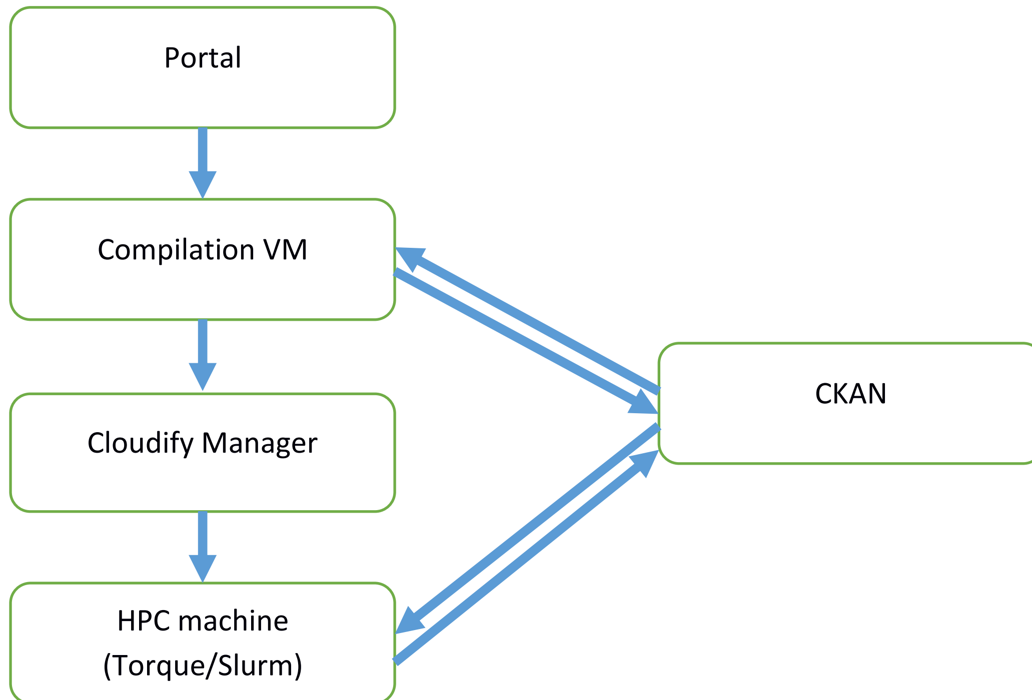

Figure 9. Integration architecture

First, the portal invokes the DSL compilation on the compilation VM, which produces low-level source code files and/or configuration files for the computational kernel, which are uploaded to CKAN. Then, a fresh blueprint is uploaded and deployed through Cloudify, which triggers a computational job on the HPC machine through the native job management system (Torque or Slurm).

Finally, the computational job fetches the program and data files, compiles the program files, and executes the resulting program on the fetched data files. The resulting output files are uploaded back to CKAN.

In order to run a DSL job using the Portal, the user first develops the DSL program on their local machine, and then generates a Cloudify blueprint, which represents the computational job. The blueprint is then uploaded using the submission page shown in Figure 10. The submission system is further described in Deliverable 5.4.

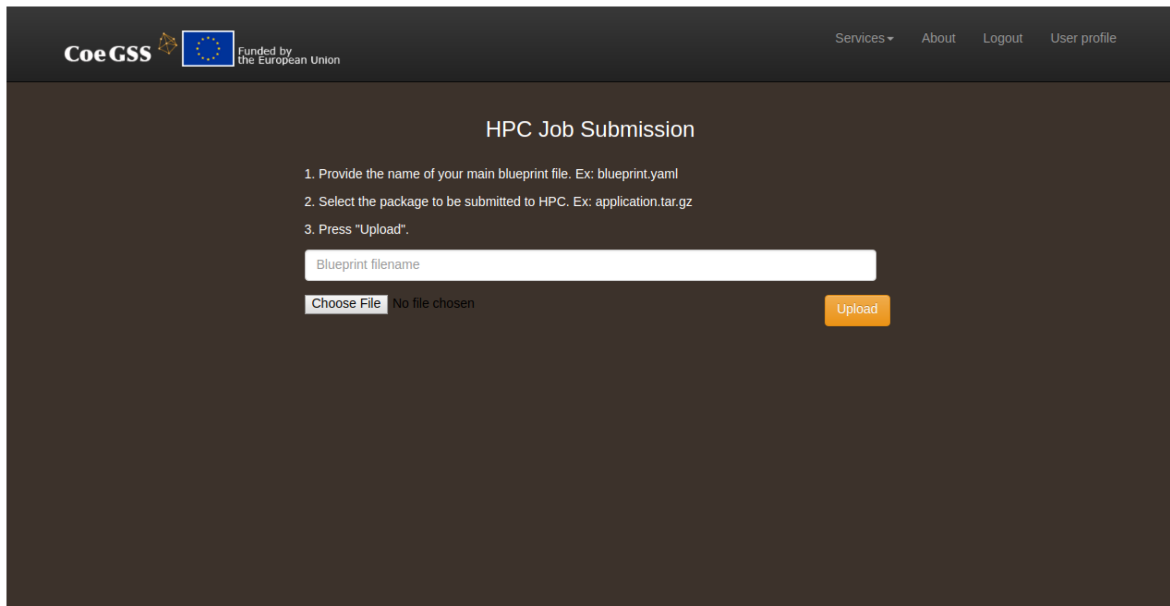

Figure 10. Submission page for HPC jobs

5.4 Synthetic Population generation: performance considerations

The performance of synthetic population generation has been evaluated on demographic data from the following sources:

- Eurostat census data (2001)[18]
- Eurostat education attainment level data (2001)[19]
- UK General Household Survey (micro-sample, 2006) [11]

Using a program implemented in DSLP, the micro-sample of 22924 records was binned into 180 categories (a combination of 6 education levels and 30 demographic categories), and then the IPF algorithm was used to fit the resulting counts to marginal data coming from the Eurostat data sets for 5 UK regions. The fitted counts are then used as weights for sampling agents from the respective buckets. Figure 11**Błąd! Nie można odnaleźć źródła odwołania.** shows the performance of the generation procedure when 10 agents are generated for each of the 5 regions, and when 10000 agents are generated for each of them.

[18] Eurostat census data http://appsso.eurostat.ec.europa.eu/nui/show.do?dataset=cens_01rapop&lang=en

[19] Eurostat education attainment level data http://appsso.eurostat.ec.europa.eu/nui/show.do?dataset=cens_01reisco&lang=en

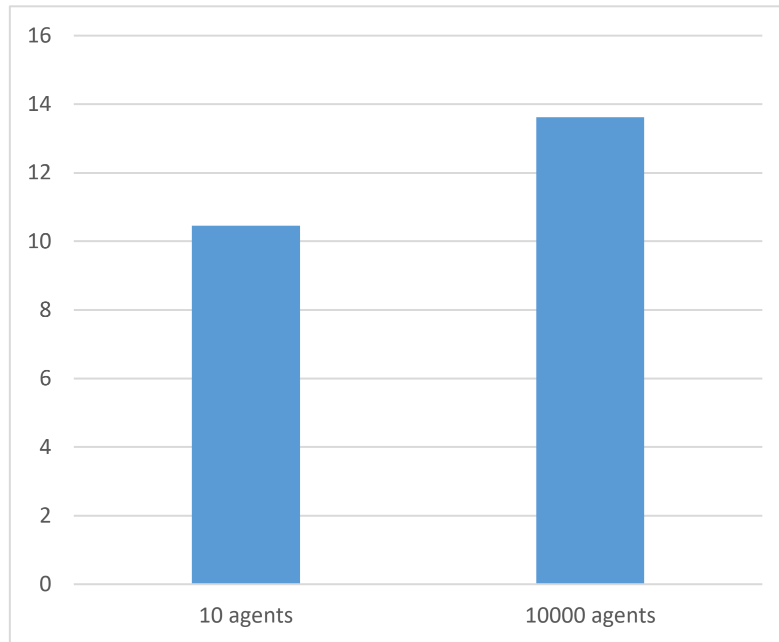

Figure 11. Performance of synthetic population generation (in seconds, median of 5 runs)

As visible in the Figure, sampling of 10000 agents takes about 25% of the run time, while the rest is taken by reading the input files and the fitting procedure. Furthermore, we have determined that I/O takes the majority of the run time, and thus in the current implementation the performance of the procedure is I/O-bound.

6 Network reconstruction tool

The dynamic of an ABM relies on the way agents are connected, i.e. in the way they influence each other through their behaviours. Representing correctly the network of interactions has thus a capital role in the ABM definition. Luckily, even if friendship relations are hard to measure (mostly they are inferred from different measures [12]), some properties seem to be ubiquitous. In particular, a hierarchical community structure is observed in all networks [12].

In CoeGSS we tackle the problem of the influence network by defining a weighted network of similarity from the attributes defining an agent and damping it through a function of the distance between the agents. As already described in the previous sections, an agent is generated by the synthetic population tool and it is equipped with several attributes (age, sex, education…). The target of the Similarity Network for Synthetic Population (SN4SP) tool is to derive an influence network from the attributes of the agents of a synthetic population.

The problem is threefold. First, we have to decide what is a similarity, secondly how to handle different kinds of data (agent attributes may be categorical, numerical or geographical) and finally we need to be sure to avoid external bias to our tool. Lin Similarity [12] successfully solves all these issues. In our realisation, we even explicitly consider possible dependencies among attributes, an issue that is often present in real data but rarely considered in the applications.

In particular, this last point, which is left implicit in the original definition of Lin, was addressed in the last months. Moreover, we tested the results even on datasets different from the ones of interest for CoeGSS and we obtained reliable results.

In order to apply this methodology to the description of actual influence relations, we introduced a damping factor depending on the distance between agents. Several distance dampings are considered: in the literature, for instance, we can find different power laws, depending on the medium implemented to infer the friendship relations.

6.1 Parallelization efforts

Important modifications were employed with respect to the previous versions. Indeed, the calculation times are affected by the dependencies present among the different attributes. In order to reduce calculation times, we sample the whole dataset and infer the value of the similarity from the results obtained on the sample.

The algorithm of SN4SP is organised in four steps. First, the code pre-processes data to feed the main algorithm in the correct way. Then a sample is realised; the user can decide the dimension of the sample. Thirdly, the distance between agents is calculated: if it is too much, the damping factor is too small to contribute significatively to the similarity and the weight value is set to zero and the calculation of the Lin similarity between the agents is neglected. However, if the distance between the agents is small enough for providing a considerable

contribution, we calculate the Lin similarity on the sample and assign it to the pair of nodes under consideration.

Actually, the only information needed is the synthetic population and the kind of attributes (categorical, numerical, and geographical). In principle, the user may decide that some attributes are redundant or not necessary for the actual calculations: this choice can be performed in the pre-processing step.

The algorithm has just two parameters, which are the distance at which the weight of the link is reduced by a factor ½ and the damping function. The dimension of the sample represents an additional parameter, related to the precision of the weight in the influence network.

The code is implemented in python, making use of the mpi4py and h5py packages, i.e. python wrappers respectively for MPI (Message Passing Interface) and the (parallelised) management of HDF5 files. The numerical steps use the Dask python package, which is optimised on clusters. The actual version is running on the HazelHen machine.

The output of the similarity network algorithm is saved in the original input HDF5 file format. An extra HDF5 dataset contains the global similarity. Actually, the results are saved in a brand new HDF5 file format with the structure defined above.

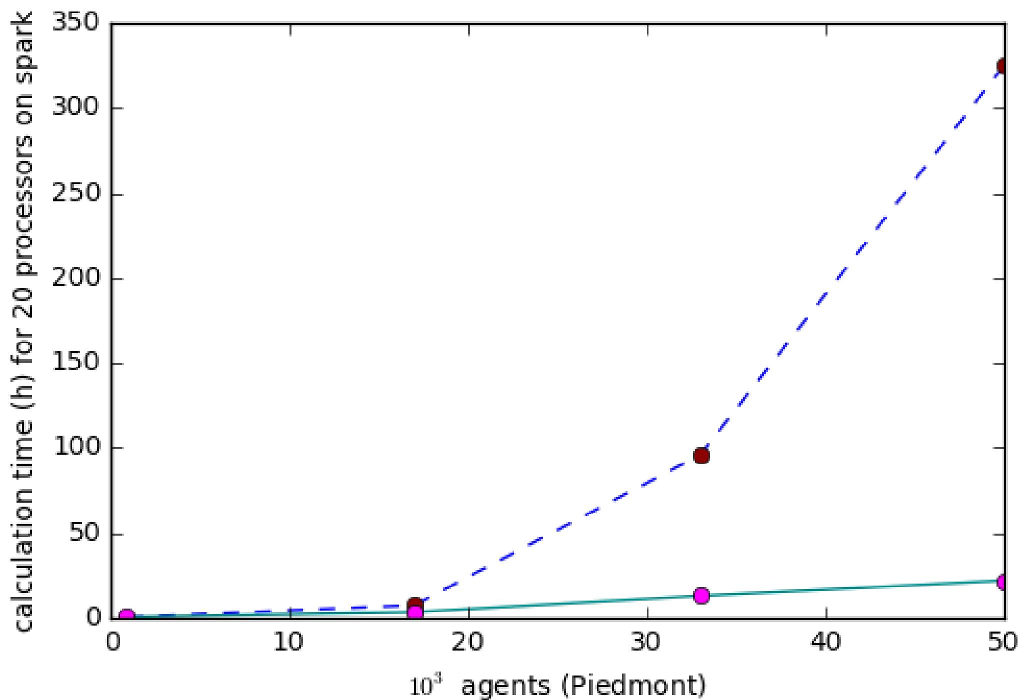

Figure 12. Scalability of Network Reconstruction tool - calculation time vs. number of agents

In the plot, the cyan line represents the calculation time, when a training set of the 10% of the whole population is used, instead of the entire population (blue dashed line). As it is possible to see, introducing the sample drastically reduces the calculation times.

6.2 Network reconstruction tool Web GUI

The portal enables launching applications on the remote HPC systems with the help of Cloudify [13]. For each application launched remotely, Cloudify requires configuration files called blueprints. Blueprint files are YAML files written in accordance with the OASIS TOSCA standard [14]. They describe the execution plans for the lifecycle of the application including installing, starting, terminating, orchestrating, and monitoring steps.

In our case, the network reconstruction algorithm is implemented as a Python script. The blueprint for this script makes use of the Cloudify HPC plugin presented in D5.11 [15].

This blueprint defines the inputs that specify arguments of the script and the inputs that specify details of the application lifecycle. The first group includes the following inputs:

- “synpop_path” which contains the path to the input HDF5 file with the synthetic population. It does not have a default value and, thus, must be specified in inputs when the blueprint is deployed.

- “synnet_path” which contains the name to the output HDF5 file with the synthesized network. By default, we use the name of the synthetic population input file suffixed with “_network”.

- “half_similarity” which defines half-similarity scale. The default value is 5000 (the value is intended to be in meters).

- “damping” which defines the damping function. If the damping value is 0, we use exponential damping, otherwise we use a power law with user-defined exponent. By default, it is set to 0 (i.e. exponential damping).

- “stripe_size” which represents percentage of the sample for the similarity calculation. The default value is 0.1.

The second group contains the following inputs:

- “hpc_configuration” which defines workload manager information and credentials of the remote HPC system to run the script on. This parameter does not have a default value and, thus, must be specified in the inputs when the blueprint is deployed.

- “num_tasks” which defines the number of MPI processes.

- “monitor_entrypoint” which specifies IP of the external task monitoring server. By default, we use the simple Python monitor implemented directly into the Cloudify HPC plugin.

- “python_module” which defines the name of the module that must be loaded on the target HPC to get access to the Python 2.7 environment with libraries that the network reconstruction script depends on. In particular, this environment must include mpi4py>=2.0, H5py>=2.8, DateTime>=4.2, psutil>=5.4.7, matplotlib>=2.2.3, numpy>=1.11.3 scipy>=0.19, pandas>=0.23.4, geopandas>=0.4. By default, we assume that the name of the module is “tools/python/coegss/2.7”.

- "job_prefix" which contains job name prefix on HPC. Default prefix is "coegss".

In order to make things run smoothly on different HPC clusters, we accompany the blueprint file with job bootstrapping and reversion bash scripts. The bootstrapping script creates batch scripts for the workload manager (TORQUE or SLURM) and allocates workspace for the network reconstruction output file, if the latter is supported by the target HPC cluster. Note that workspace allocation helps to avoid IO problems since the output files require $O(n^2)$ disk space and may easily overcome limits for the user's home folder. The reversion script takes care of uploading output files to CKAN, releases the workspace, and removes the batch script for the workload manager.

In order to run the application with the Cloudify manager, the user must do the following steps: upload the blueprint to the manager, deploy this blueprint, launch "install" and "job_run" executions. Since the network reconstruction script is a part of the CoeGSS toolchain, its blueprint is already uploaded to our Cloudify manager VMs manually. So, in contrast to the other steps, the blueprint-uploading step does not require interaction with the portal. For the remaining three steps, the portal interacts with the Cloudify managers via the Cloudify REST client API [16]. In particular, in the blueprint deployment step, we convey blueprint inputs from the portal Web GUI to the Cloudify manager and the Cloudify manager copies application bootstrapping and reversion scripts to the target HPC cluster. After the blueprint deployment, we run the "install" execution, which launches bootstrapping scripts with the blueprint inputs as arguments. As a result, we obtain a batch script for the workload manager. Finally, we put this batch script into the workload manager queue via calling the "job_run" execution in the Cloudify manager. As soon as the parallel job is finished and the portal gets informed about it by the Cloudify manager, the portal calls the "uninstall" execution, which launches the job reversion script. Afterwards, the portal deletes the blueprint deployment.

7 Representing Uncertainty in Modelling and Computation

In the following section, we describe the packages and codes that have been developed in Task 3.2 during the lifetime of the project. All codes are contained in the CoeGSS code repository on GitHub.

7.1 Description of Software Packages

Intervals: is a Haskell package that contains the data types interval and interval lists, arithmetic operations *add, sub, mul, divsimple, div* to be performed on these data types. The implementation follows the IEEE1788 standard for interval mathematics. To ensure that the resulting intervals are always safe, we worked with the standard Haskell rounding mode (rounding to the nearest) and perform the "outwards" rounding explicitly as described in [17]. With *simpleDiv* as fourth operation, the arithmetics is a closed calculus on intervals that throws an exception in case that the divisor interval contains zero.

With division operation, *div* as the fourth operation, the calculus has lists of intervals as basic data type. If one tries to divide by an interval containing zero, then the result is a pair of intervals. Dividing a singleton list of intervals by the singleton list [[0,0]] leads to the empty list as a result.

The IPF procedure that is used within the project for the creation of synthetic populations was validated by Task 3.2. The Haskell package **IPF-Ground-One** contains all necessary modules for it. As a proof of concept for our approach, it shows how to use the interval library to validate an arbitrary numerical procedure. The example was interesting not only because the algorithm is one that is directly used in the project but also because it shows how the method is a successful tool to evaluate the precision without having to perform a thorough numerical analysis. IPF is an iterative procedure that in theory is proven to converge. While converging is a notion that makes sense only for real numbers, when working with floating point numbers, the best thing one can hope for is that a fixed point is reached after several iterations. Our interval extension of two dimensional and n-dimensional (n<=7) IPF showed that after 6-8 iterations the changes in the result are rather small. With more iterations, the rounding errors get an increasing influence on the result. The produced matrices contain intervals with increasing size, i.e. the imprecision or uncertainty of the result is increasing. The approach for validation of a numerical procedure at the example of the IPF algorithm was presented in a talk, the 2018 Workshop on Numerical Programming in Functional Languages.

To generate random test samples for input matrices we implemented a particular module **SampleGen** and added it to the **IPF-GroundOne** package. After choosing a range for the values of the marginals and the matrix entries, inputs follow an equal distribution. Also, a graphics module **IPFGraphResults** was created to visualise the behaviour using LaTeX TikZ.

As an additional tool to validate numerical procedures, we implemented an interval version of the Newton method for finding zeros and the midpoint method to find extrema of a given function (modules **Newton** and **MidPointMethod**) described in [18]. They expect an interval extension of said function and its first derivative as an input, but the modules **Polynomial** and **Expression** may be used to create these functions for certain special cases. Not all interval realisations of the functions used in **Expression,** however are (safe) interval extensions. In their current state, they were only used for testing purposes of the algorithms. Both methods yield as a result an interval or a list of intervals that contains the (real valued) “solution”, if certain conditions are met: the input has to fulfil the input condition (for example for Newton the derivative is not allowed to be zero in the input interval), there must exist a solution for the considered input and the input functions have to be proper interval extensions. Calling the algorithms with a function from **Expression** is not "safe" in this sense, although it should still be relatively accurate. All codes are contained in the package **Newton-Algorithm-and-Midpoint-Method-for-Intervals.**

In the very beginning of the project as preliminary work for the optimisation procedures we expected to validate, we implemented an Idris module for the divide and conquer algorithm scheme based on [19] that was later on translated into Agda, which turned out to be the language that better fits the purpose. Divide and Conquer is a well-known algorithm scheme that is very suitable to synthesise provably correct algorithms. Our Agda implementation of it is thus not only a template for the implementation of a certain instance of it (like merge sort or branch and bound search) but guarantees also that if the user instantiates its ingredients (input and output conditions, split and compose functions and a solve-directly function on primitives) and instantiates the axioms, the resulting implementation contains its correctness proof. The instantiation is done by filling in the unimplemented elements ("*holes*") in these files. They are automatically discovered and highlighted by loading the files in an Agda-aware editor such as Emacs. Examples of implementing a Divide and Conquer algorithm are found in the modules **Mergesort** and **Quicksort** contained in the **Divide-and-Conquer-in-Agda** package [20].

The method can be used to design and implement for example optimisation algorithms, which are important for model adaptation and parameter estimation in GSS applications.

8 Hardware and software co-design

In this chapter, we describe our recent findings related to software-software and hardware-software co-design. In particular, as a part of software-software co-design discussion, in section 8.1, we present benchmarking results for the HDF5 extension library [9]. Section 8.2 addresses aspects related to hardware-software co-design. In this section, we formulate general recommendations for hardware providers that might help to build clusters targeting GSS users. Those recommendations are based on the extensive benchmarking of eight HPC applications coming from various domains relevant for GSS (such as ABMs in social sciences, simulation of pollutions, etc.) on nine different recently emerged architectures. The results of benchmarking were previously reported in the deliverables D5.7 [21] and D5.8 [22].

8.1 Performance of the HDF5 extension library

In D3.4, we presented the HFD5 file structure, which can be used by different software components of the CoeGSS toolchain and an HDF5 extension library that facilitates manipulations with HFD5 files that follow this structure. In order to assess performance of this HDF5 extension library, we prepared a benchmark that implements Axelrod's model of dissemination of culture with the help of the Amos framework. This model was proposed in 1996 by R. Axelrod [23] and immediately gained broad popularity among social scientists. Nowadays, it is considered as one of the most well studied ABMs – both theoretically and empirically [24], – which motivated us to choose Axelrod's model for benchmarking.

The model defines agents and rules for their interactions as follows. Agents model individuals in a culture dissemination process. Each agent is endowed with F integer attributes called cultural traits, which are meant to model different beliefs, opinions, and other properties of agents. The model allows only a limited number of values for each cultural trait $f_i = \{0,1,\ldots,q_i - 1\}$. In the dynamic step, each agent randomly selects one neighbour and the agent interacts with the neighbour with some probability proportional to the overlaps between the agent-neighbour pairs (the overlap is computed as a number of equal features). The interaction consists in assigning to one of the agent’s trait the value of its neighbour’s trait. In other words, these rules make interacting agents more similar, but the interaction happens more often if agents already share many traits and it never happens if agents have no trait in common. This suggests that Axelrod's interaction rules allow to model two cultural mechanisms – social influence and homophily. In order to fit Axelrod’s model to the graph-based ABMS framework discussed in D3.3, we slightly modified Axelrod’s notion of neighbours. In our implementation, we consider as neighbours all agents located at the same spatial site, as well as the agents that have direct links in the social graphs.

The benchmark was performed on the Hazelhen cluster at HLRS. Hazelhen is composed of CRAY XC40 nodes and has peak performance 7.4 Pflops. The cluster includes 41 Cray cascade cabinets with in total 7712 dual socket compute nodes. Each node is equipped with 2 12-core Intel Haswell E5-2680v3 CPUs and 128GB of DDR4 RAM. The type of interconnect is Cray Aries.

It uses Lustre storage and operates with the Cray Linux Environment. We compiled all components – Amos, HDF5 extension library, and codes of Axelrod's model – with GCC 6.4. Then the HDF5 extension library was linked against HDF5v1.10.2.

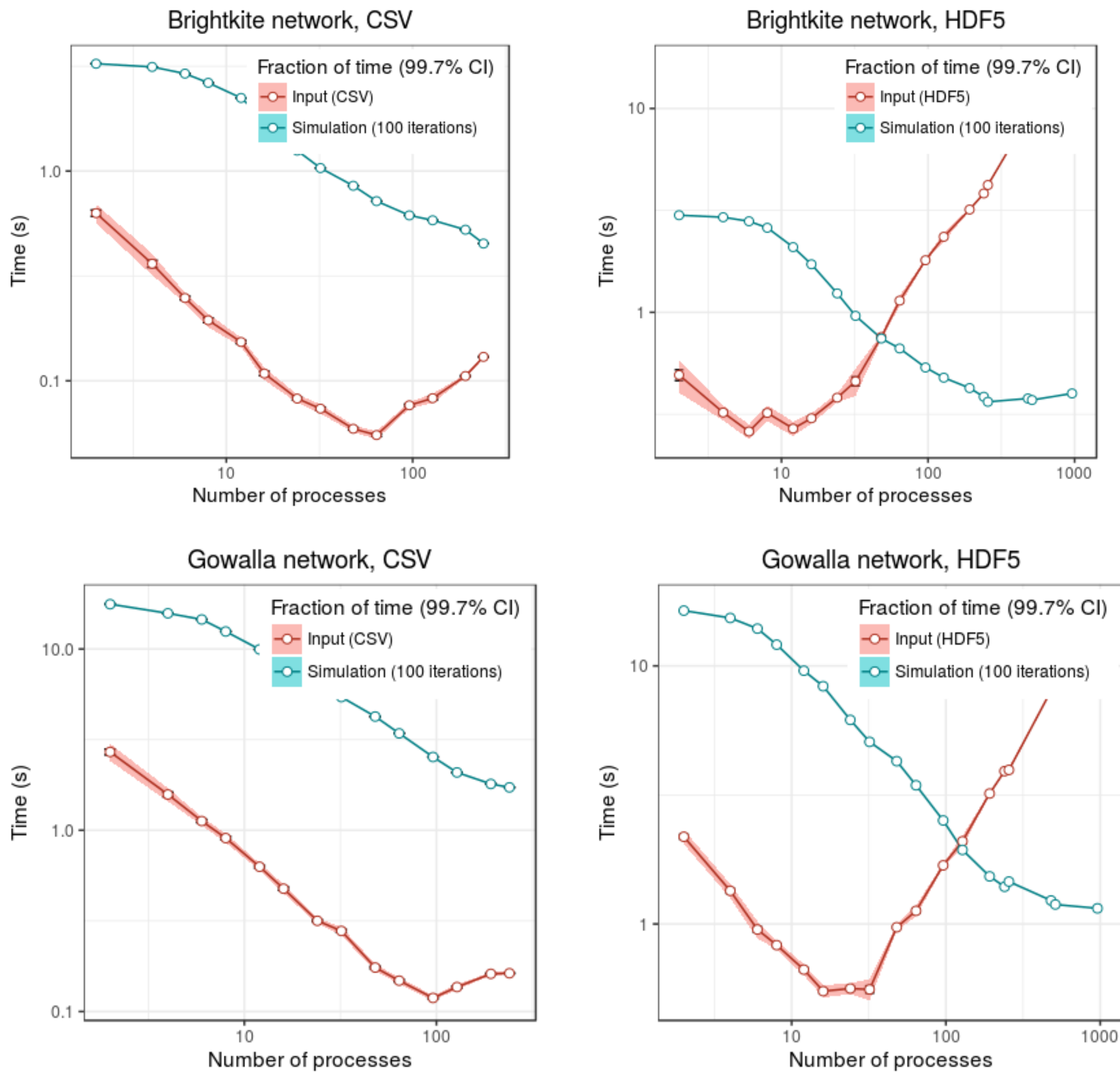

Figure 13. Scalability of Axelrod's model implemented with the Amos framework and the CoeGSS HDF5 extension library on Hazelhen cluster

In our benchmarks, we used 2 networks from – Brighkight and Gowala – for representing long-range interactions. The number of agents was artificially adjusted to the number of vertices in the networks. In order to create sites for their allocation, we used a 240x290 pixel raster with a population density heat map for the Faroe islands from Eurostat. We initialized agents with three cultural features each taking random values between 0 and 9.

Figure 13 summarizes results of the benchmark. Its subplots contain line chart with confidence intervals for measured elapsed times of data input and 100 iterations of Axelrod's interaction iterations. In these plots, we compare performance of the HDF5 extension library against

embarrassingly parallel CSV input on both social networks. By embarrassingly parallel CSV input, we mean a naïve embarrassingly parallel implementation of the CSV reader for Amos which assumes that the user split the input data and prepared CSV files for each MPI process separately. For both networks, the scalability of the embarassingly parallel CSV input is much better than the scalability of the HDF5 extension library. After reaching the scalability limit, elapsed time of the HDF5 extension library increases dramatically. The latter makes us believe that performance of the HDF5 extension library can be significantly improved if it were implemented in a way that a relatively small number of the processes reads the HDF5 files and distributes the data between processes.

8.2 Analysis of CoeGSS benchmarks from the hardware/software co-design perspective

In the previous deliverables, we covered many aspects of software/software co-design while our hardware/software co-design findings were not covered at an appropriate depth. The following text aims to reduce this imbalance. During the project lifetime, we benchmarked a number of diverse HPC compliant GSS related applications on various recently released HPC platforms. In this section, we do not present our benchmarking results, but rather draw recommendations for the HPC hardware vendors based on those results. We refer the readers interested in the benchmarking results to our deliverables D5.7 and D5.8.

Functionally, the benchmarked applications can be categorized into two groups: social simulation software and large-scale CFD (Computational Fluid Dynamics) applications. From the perspective of programming languages, our benchmarks covered applications written in C++ and Python, which are the most popular programming languages among GSS experts who use HPC.

The group of benchmarked social simulation software covers applications for pre-processing and simulation of agent-based models. We considered two typical pre-processing tasks: converting shape files to rasters and producing synthetic populations. The benchmarks have shown that conversion from shape files to rasters is very demanding in terms of RAM while I/O and CPU requirements are rather low. In order to study computational requirements to applications that generate synthetic populations, we implemented a simple parallel version of the celebrated iterative proportional fitting (IPF) method. Our implementation heavily uses dense linear algebra kernels provided by a highly optimised ScaLAPACK library. Our benchmarks demonstrate high performance of IPF on different architectures. Neither RAM, nor I/O of modern architectures are limiting factors for IPF performance. Along with ABMS pre-processing applications, we benchmarked a simple agent-based model of diffusion with the help of distributed ABMS frameworks following two different parallelization strategies for ABMs with raster inputs. The first framework – Pandora – is written in C++ and parallelizes the simulation process via splitting of rasters on even pieces and distributing them between MPI processes, while the second framework – ABM4py – is a Python code, which implements the graph-based parallelization approach discussed in D3.3. Despite strong difference in

parallelization strategies, in both cases, we observe the same pattern: ABMS applications produce a big amount of output which has a strong negative impact on application performance. As a consequence, according to our toy ABMs, being I/O bound, current ABMS frameworks for HPC have moderate requirements to CPU performance. Nevertheless, we must emphasize that the results can look differently for complex models with sophisticated agent activities and models which can be reduced to iterations with sparse-matrix dense-vector operations, thus, our benchmarks for ABMS frameworks are not very illuminative and must be extended with more sophisticated models to draw stronger conclusions. But discussion of the new ABMS models for benchmarking goes beyond the scope of this text. Table 2 shortly summarizes information about scalability of the benchmarked social simulation software and hardware bottlenecks.

		preprocessing		ABMS (with raster input)		
		restering	IPF	Pandora		ABM4py
				Europe	World	128x128
Bottlenecks	CPU		+			
	RAM					+
	IO			output	output	output
	Network	N.A.				
	Scalability*	serial	≥ 1400	≈ 128	≈ 700	≈ 128

* maximum number of utilized cores of Xeon E5 2682 v4 cluster that help to reduce a total elapsed time

Table 2. Bottlenecks in the hardware and scalability for the application from social simulation software stack according to the CoeGSS benchmarks

The group of benchmarked CFD applications includes large scale tools that simulate GSS-related scenarios like natural disasters (hurricanes, earthquakes), spread of air pollutions, and weather prediction. More specifically, we selected the following open-source CFD codes for our benchmarks:

- HRWF – a parallel implementation of the hurricane weather research and forecasting (HWRF) model [21];
- OpenSWPC – an integrated parallel simulation code for modelling seismic wave propagation in 3D heterogeneous viscoelastic media [22];
- CMAQ – a community multiscale air quality modelling system, which combines CFD codes for conducting large scale air quality model simulations [25];
- CM1 – a parallel implementation of the three-dimensional, time-dependent, non-hydrostatic numerical model for studies of small-scale processes in the Earth’s atmosphere, such as thunderstorms, etc. [24]

As expected, our benchmarks confirm that CFD applications are in general CPU-bound in contrast to the social simulation software. Nevertheless, we observed that at some architectures' memory was also a bottleneck for some choices of the number of MPI processes. In particular, CMAQ (Community Multiscale Air Quality) and CM1 applications produce a lot of outputs, which imposed additional performance constraints on architectures with poor I/O speed. In addition, we noticed that OpenSWPC (Open-source Seismic Wave Propagation Code) is memory bound for the small number of MPI processes. We outlined relevant information about scalability of the benchmarked CFD applications and hardware bottlenecks in Table 3.

		HRWF (hurricanes)	OpenSWPC (seismology)	CMAQ/CCTM (pollutions)	CM1 (weather)
Bottlenecks	CPU	+	+	+	+
	RAM		+		
	IO	input		output	output
	Network				
	Scalability*	≥ 128	≥ 128	≥ 128	≥ 128

* maximum number of utilized cores of Xeon E5 2682 v4 cluster that help to reduce a total elapsed time

Table 3. Bottlenecks in the hardware and scalability for the application from large-scale CFD applications according to the CoeGSS benchmarks

As Tables 2-3 illustrate, most of the distributed GSS applications are memory bound. Even large-scale CFD codes can be bound by I/O and RAM under special circumstances. It allows us to conclude that the fast memory is an essential requirement to HPC clusters for GSS applications whereas high CPU's clock frequency plays a less important role. Moreover, since many state-of-the-art GSS applications deal with large input and output files, we believe that GSS software developers should invest more time into designing file-avoiding applications. Our scalability tests show that hyperthreading provides little performance improvements for most of the GSS applications. Therefore, it makes little sense to invest money in expensive massively multithreaded chips (like Power8) for GSS users. We also recommend avoiding clusters with GPU accelerated nodes since only few popular GSS applications benefit from GPUs. In particular, among widely used general-purpose ABMS frameworks and problem-specific ABMS codes for HPCs, only the FLAME-HPC framework utilizes GPUs. Weak use of GPUs is also partially related to the fact that most social science applications are memory bound. Being more specific, among the architectures used in benchmarking, we recommend to build clusters upon ARM Hi1616 if energy efficiency is a crucial requirement, or upon Intel® Xeon® Gold 6140 if performance is a first priority while relatively high operating expense and capital expenditure are not an issue.

According to our benchmarks, the scalability of GSS applications is rather diverse. All applications from the social simulation software stack demonstrate poor scalability with one notable exception – the IPF implementation. Moreover, even though our benchmarks do not demonstrate this explicitly, it is also known that social simulation software scales worse than the large-scale CFD codes. On the other hand, due to stochastic nature of ABMs, a typical social simulation workflow assumes many simultaneous simulation runs, whereas the fitting step in reconstruction of a synthetic population should normally be performed only once for a given dataset. Therefore, the optimal number of nodes for the state-of-the-art should be defined by scalability of the synthetic population and CFD codes (if the latter are of interest for the target GSS audience). We can always bypass the gap in scalability of the synthetic population and ABMS codes and reach full utilization of clusters by making several simultaneous simulation runs (and treating simulation results in a file-avoiding way). Unfortunately, our results do not allow to draw conclusions about node interconnects since most of the benchmarks were done on the testbeds with only one or two nodes.

9 Portal – HPC interoperability

As a way to facilitate the execution of simulations, the CoeGSS Portal includes a new GUI connected to an Orchestrator, so researchers do not need to take care of the deployment and usage of HPC resources. The complexity of dealing with these resources is hidden, so they can focus on the configuration and execution of the simulations. They will not need to deal with the workload managers APIs, the queues and the continuous execution of complex scripts.

9.1 Running Simulations Through the Portal

In the previous release of the CoeGSS Portal, the HPC Orchestrator component was introduced as a component, which receives a workflow specification (in a language called TOSCA [14]) and is able to send jobs to the corresponding HPC centres. Such Orchestrator is based on Cloudify, with specific plug-ins for enabling the usage of Slurm and Torque, as a way to send jobs to large HPC systems.

The usage of such orchestrator enables the possibility to use TOSCA for implementing simulation workflows in the context of CoeGSS. It is possible to specify a specification for running stand-alone tools (e.g. the synthetic population generation tool), so single runs of the tool with different configurations can be made. On the other hand, it is possible to implement the complete CoeGSS workflow by including all the necessary tasks in the TOSCA file, with the corresponding scripts.

When pilots want to run a complete simulation, they specify all the phases of the workflow within TOSCA, specifying the jobs to run and their characteristics (as explained in the following subsection). This means that it will include jobs for data pre-processing and movement (if necessary), jobs for population generation and network reconstruction, jobs for launching several simulations and jobs for the post-processing.

It is necessary that developers prepare all the TOSCA files and that they include all the scripts that will be executed, according to the TOSCA specification. Inputs and outputs are already specified for each task, so it is possible to parametrize the simulation. Finally, everything is packaged in a tar file that can be processed through the CoeGSS Portal.

The way to proceed for running a simulation by a user consists in the following steps:

1. Upload the data to a repository enabled by CoeGSS and register such input data in CKAN (accessible through the CoeGSS Portal);
2. Open the TOSCA file and change any input parameter as required (certain users may want to change the number of nodes to use, the number of parallel simulations to run, etc.), saving any change done in the simulation package file;
3. Enter the jobs submission interface in the CoeGSS Portal;
4. Click the ‘Choose File’ button and select the package to execute by using the Portal interface (that is, the application package with the blueprint and scripts, as given by the developers);

5. Click the second 'Choose File' button and select the file with the input parameters for the selected applications;
6. Click on 'Run'.

The CoEGSS Portal will retrieve the specification and it will communicate with the Orchestrator to launch the simulation workflow, retrieving the data, moving it adequately and executing all the tools included in such workflow. At the end of the process, the resulting data will be stored as specified in the TOSCA and/or input files and the user will be notified.

9.2 Specification of the Simulations with TOSCA

TOSCA is a standard specification for workflows, which facilitates the deployment of applications in Cloud environments. It is considered a DSL and, since it was not designed for supporting HPC environments, some changes were proposed in [26]. These changes, basically, were oriented to support the execution of mathematical simulations in a hybrid environment where HPC and Cloud resources are available. CoEGSS has taken advantage of such extensions, in order to create the simulation workflows to be implemented in the project. The definitions of such workflows are considered blueprints in the context of Cloudify (which is the base software of the Orchestrator component).

The extensions of the DSL specify two main types that must be used in the workflow specification: Compute and Job.

Each Compute defines an HPC centre that is providing computational resources through the Orchestrator. It is used for specifying the entry point (URL, port, etc.), the workload manager used (since it will be necessary to use one connector or another), user's credentials for accessing the resources and other parameters for configuring the monitoring, the working directory and additional configuration.

In the case of CoEGSS, we have defined Compute elements for PSNC and HLRS infrastructures. This is done after the block which defines the input variables of the workflow.

```
47  node_templates:
48      first_hpc:
49          type: hpc.nodes.Compute
50          properties:
51              config: { get_input: coegss_hlrs_laki }
52              # external_monitor_entrypoint: { get_input: monitor_entrypoint }
53              job_prefix: { get_input: job_prefix }
54              base_dir: "$HOME"
55              workdir_prefix: "single_sbatch"
56              skip_cleanup: True
```

Figure 14. Definition of the Compute element for the HLRS infrastructure

Each Job represents a task to be carried out in the workflow. It has three main phases:

- Bootstrap: it represents tasks to be performed before running the task (e.g. move some data);

- Execution: it runs a concrete application or task, as defined in the workflow;
- Revert: it carries out tasks that must be done after the main task has finished (i.e. do some action on the resulting data, move the output data to certain location, etc.).

Usually, in the bootstrap and revert part, developers just indicate the location of the script to be executed, so the script may perform several activities. In the case of the main action, it is possible to determine the inputs and outputs involved, the amount of resources to use (i.e. number of nodes), modules that must be loaded for running the task, maximum time for running the task, whether several tasks will be run in the same node (related to scalability) or even the type of task (batch or normal job).

```
58  single_job:
59      type: hpc.nodes.job
60      properties:
61          job_options:
62              type: 'SBATCH'
63              command: "touch.script"
64          deployment:
65              bootstrap: 'scripts/bootstrap_sbatch_example.sh'
66              revert: 'scripts/revert_sbatch_example.sh'
67              inputs:
68                  - 'single'
69          skip_cleanup: True
70      relationships:
71          - type: job_contained_in_hpc
72            target: first_hpc
73
```

Figure 15. Definition of a single_job

It is important to highlight that the definition of inputs and outputs follow a simple structure, with a description, the type of input (integer, string, float…) and a default value (if we want to set one).

```
31  # Job prefix name
32  job_prefix:
33      description: Job name prefix in HPCs
34      default: "coegss"
35      type: string
36
```

Figure 16. Example of input variable

When running the workflow, all the inputs must be included in a separate file, which will be assigned to the workflow with the corresponding command. Such file will just include pairs of values (*name_of_input_variable: value_of_variable*).

10 Summary

In the deliverable, we presented the status of tools and methods in development that were implemented in the final release of the portal. It also presents other achievements of WP3 tasks like increasing software execution reliability, remote visualization using the COVISE system. Moreover, we discussed different implementations of DSLs for different approaches in the project. Next, we presented achievements in parallelization of the network reconstruction tool as well as packages and codes for representing uncertainty in modelling. In the co-design section, we focused on software-software and hardware-software co-design in two aspects: performance of the HDF5 extension library and benchmarking HPC compliant GSS related applications. At the end we presented how Portal-HPC interaction is organized in respect of launching simulations from the portal and how submitted tasks are described in a TOSCA specification.

All presented tools, methods and solutions implemented for the CoeGSS project are very useful in Global Systems Science processing in general and can be adapted for other use cases. The results presented like software functionality provided and scores achieved in parallelization of tools are satisfactory and provide much substantial information for CoeGSS potential users and similar system developers.

References

[1] "DMTCP project home page," [Online]. Available: http://dmtcp.sourceforge.net/.

[2] "Dakota," [Online]. Available: https://dakota.sandia.gov/.

[3] "Cloudify," [Online]. Available: https://cloudify.co/.

[4] "R package Shiny," [Online]. Available: https://shiny.rstudio.com/.

[5] C. Ionescu, "Vulnerability Modelling with Functional Programming and Dependent Types," *Mathematical Structures in Computer Science 26 (01). Cambridge University Press: 114–28. ,* no. doi:10.1017/S0960129514000139., 2016.

[6] N. B. e. al., "Sequential decision problems, dependent types and generic solutions," 2017.

[7] N. Botta, P. Jansson and C. Ionescu, "Contributions to a Computational Theory of Policy Advice and Avoidability," no. doi:10.1017/S0956796817000156, 2017.

[8] N. Botta, P. Jansson and C. Ionescu, "The Impact of Uncertainty on Optimal Emission Policies," *Earth System Dynamics 9 (2): 525–42.,* no. doi:10.5194/esd-9-525-2018., 2018.

[9] Office for National Statistics. Social and Vital Statistics Division, "General Household Survey, 2006, 3rd Edition," UK Data Service, 2009.

[10] S. Najd, S. Lindley, J. Svenningsson and P. Wadler, "Everything old is new again: Quoted Domain Specific Languages," in *Partial Evaluation and Program Manipulation 2016*, St. Petersburg, 2016.

[11] "General Household Survey, 2006 [computer file]. 3rd Edition. Colchester, Essex: UK Data Archive [distributor]," *February 2009,* no. SN: 5804, February 2009.

[12] D. Lin, "An Information-Theoretic Definition of Similarity," in *ICML '98 Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[13] "Cloudify documentation," [Online]. Available: https://docs.cloudify.co/4.4.0/.

[14] "Topology and Orchestration Specification for Cloud Applications Version 1.0 OASIS Standard, 2013, 114 p.," [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf.

[15] F. Nieto, "D5.11 - Second Portal Release.".

[16] "`cloudify-rest-client`'s documentation.," [Online]. Available: https://cloudify-rest-client.readthedocs.io/en/3.2/.

[17] S. M. Rump, T. Ogita, Y. Morikura and S. Oishi, "Interval arithmetic with fixed rounding mode," *Nonlinear Theory and Its Applications, IEICE vol. 7(3),* no. doi 10.1587/nolta.7.362., pp. pp 362-373, 2016.

[18] W. Tucker, "Validated Numerics: A Short Introduction to Rigorous Computations," *Princeton University Press,* 2011.

[19] D. Smith, "The design of divide and conquer algorithms.," *Science of Computer Programming. 5.,* no. 10.1016/0167-6423(85)90003-6., pp. 37-58., 1985.

[20] E. Richter and M. Richter, "Error analysis almost for free," in *Workshop Numerical Programming in Functional Languages*, 2018.

[21] M. K. Biswas and e. al., "Hurricane Weather Research and Forecasting (HWRF) Model. Scientific Documentation," *NCAR Technical Note NCAR/TN-544+STR,* p. 111 pp, 2018.

[22] T. Maeda, S. Takemura and T. Furumura, "OpenSWPC: an open-source integrated parallel simulation code for modeling seismic wave propagation in 3D heterogeneous viscoelastic media," *Earth, Planets and Space,* 2017.

[23] R. Axelrod, "The Dissemination of Culture," *Journal of Conflict Resolution,* 1997.

[24] G. H. Bryan, "The governing equations for CM1. Version 4," *National Center for Atmospheric Research, Boulder, Colorado,* p. 15p.

[25] K. W. Appel and e. al., "Description and evaluation of the Community Multiscale Air Quality (CMAQ) modeling system version 5.1," *Geosci. Model Dev., 10,,* pp. p. 1703-1732, 2017.

[26] J. Carnero and F. J. Nieto, "Running Simulations in HPC and Cloud Resources by Implementing Enhanced TOSCA Workflows," in *HPCS'18*, Orleans, 2018.

[27] E. Brady, "Type-Driven Development with Idris," *Manning Publications Company.,* 2017.

[28] "Office for National Statistics. Social and Vital Statistics Division 2009," [Online]. Available: http://www.netlib.org/utk/papers/scalapack/node9.html.

[29] C. Ionescu, "Vulnerability modelling with functional programming and dependent types," 2016.

[30] N. Botta, P. Jansson, C. Ionescu, D. R. Christiansen and E. Brady, “Sequential Decision Problems, Dependent Types and Generic Solutions.,” *Logical Methods in Computer Science 13 (1).,* no. doi:10.23638/LMCS-13(1:7)2017, 2017.

[31] N. Botta, P. Jansson and C. Ionescu, “The impact of uncertainty on optimal emission policies,” 2018.

[32] N. Botta, P. Jansson and C. Ionescu, “Contributions to a computational theory of policy advice and avoidability,” 2017.

List of tables

Table 1. Checkpointing software overview 9
Table 2. Bottlenecks in the hardware and scalability for the application from social simulation software stack according to the CoeGSS benchmarks 42
Table 3. Bottlenecks in the hardware and scalability for the application from large-scale CFD applications according to the CoeGSS benchmarks 43

List of figures

Figure 1. The graph presents the AVG matrix multiplication time for C++/MPI 11
Figure 2. The graph presents the AVG matrix multiplication time for C++/OpenMP............. 12
Figure 3. The graph presents the AVG matrix multiplication time for Python/MPI 12
Figure 4. The COVISE ReadPandora module 20
Figure 5. The COVISE GetSubset module 20
Figure 6. Typical setup of the RWCovise module... 21
Figure 7. Distributed processing within COVISE.. 22
Figure 8. The TableUI standard view.. 23
Figure 9. Integration architecture 30
Figure 10. Submission page for HPC jobs 31
Figure 11. Performance of synthetic population generation (in seconds, median of 5 runs) 32
Figure 12. Scalability of Network Reconstruction tool - calculation time vs. number of agents 34
Figure 13. Scalability of Axelrod's model implemented with the Amos framework and the CoeGSS HDF5 extension library on Hazelhen cluster.. 40
Figure 14. Definition of the Compute element for the HLRS infrastructure........................... 46
Figure 15. Definition of a single_job 47
Figure 16. Example of input variable.. 47