INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelor's Thesis

Performance improvement of pre-defined HMR workflows in DRIHM

Maximilian Höb

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelor's Thesis

Performance improvement of pre-defined HMR workflows in DRIHM

Maximilian Höb

Supervision:	Prof. Dr. Dieter Kranzlmüller
Advisor:	Dr. Nils gentschen Felde
	Dr. Michael Schiffers
Date:	August 25th, 2015

Abstract

Hydro-meteorological calculations are one of the most complex computational tasks in high performance computing today. One of the biggest challenges for this science department is to predict flood events and thus to contribute to the protection of the population. DRIHM (Distributed Research Infrastructure for Hydro-Meteorology) is an EU-funded research project to combine Hydro-Metrological Research (HMR) and Information and Communication Technology (ICT) objectives in order to model weather events and its influence on suburban regions.

This thesis gives an understanding of the project, especially the requirements of the hydro-meteorological science and its embedment in a supercomputing infrastructure. Several HMR models are integrated into DRIHM. They can be configured and executed in the DRIHM front end, a web portal, and directly in the back end via command line. However the back end offers no possibility to compose models to a workflow, in contrast to the portal, which allows compositions of up to three models.These workflows lead to performance problems when submitted to grid resources because they are no real workflows but single model computations in the back end. It will be pointed out that this chaining in the portal is very inefficient and therefore very expensive with regard to computing time and transferred data.

To prevent increasing costs, lost time due to unnecessary data transfers and avoidable queueing time a concept of a complete workflow in one submission was developed. Adapted from this concept, three demonstration chains (demoChains) were added to the model repository. The first one is modularized and can handle all possible compositions of HMR models. Therefore configuration and input files were created to guarantee a smooth execution of every given workflow. The two other chains are pre-defined and cover the most important models within DRIHM.

The thesis' concept offers a framework to improve the performance of workflow executions on any back end resource. Integrated into the web portal, it could give inexperienced users a better understanding of the possibilities of chaining models within the DRIHM project.

Kurzfassung

Hydrometeorologische Berechnungen sind eine der komplexesten Berechnungsaufgaben im heutigen High Performance Computing. Eine der größten Herausforderungen für diese Wissenschaft ist es, Hochwasserereignisse vorherzusagen und damit einen Beitrag zum Schutz der Bevölkerung zu liefern. DRIHM (Distributed Research Infrastructure for Hydro-Meteorology) ist ein EU-finanziertes Forschungsprojekt, welches die Ziele der Hydro-Metrologischen Forschung (HMR) und der Informations- und Kommunikationstechnologie (ICT) kombiniert, um Wetterereignisse und deren Einfluss auf suburbane Regionen zu modellieren.

Diese Arbeit stellt das Projekt, vor allem im Hinblick auf die Anforderungen der hydrometeorologischen Wissenschaft an und ihrer Einbettung in eine Höchstleistungsrechner-Infrastruktur vor. Mehrere HMR-Modelle sind in DRIHM integriert. Sie können im DRIHM Frontend, einem Web-Portal konfiguriert und ausgeführt werden, aber auch direkt im Backend über die Kommandozeile. Allerdings bietet das Backend keine Möglichkeit, Modelle zu einem Workflow zu kombinieren. Im Gegensatz dazu ermöglicht das Portal das Verketten von bis zu drei Modellen. Aber die Abarbeitung dieser Kompositionen im Backend führt zu Leistungsproblemen, da es sich hier nicht um echte Workflows handelt, sondern um einzelne Modelle, die an die Grid-Ressourcen geschickt werden. Es wird herausgestellt, dass diese Verkettung im Portal sehr ineffizient ist und damit sehr teuer hinsichtlich der Rechenzeit und der übertragenen Daten.

Um steigenden Kosten, verlorener Zeit durch unnötige Datenübertragungen und vermeidbaren Wartezeiten vorzubeugen wurde ein Konzept eines kompletten Workflows als ein Modell entwickelt. Auf diesem Konzept basierend wurden drei Demonstrationsketten (demoChains) in die Modelle aufgenommen. Die erste ist eine modularisierte, die alle möglichen Zusammensetzungen der HMR-Modelle verarbeiten kann. Dafür wurden Konfigurations- und Inputdateien erstellt um eine reibungslose Abarbeitung jeglicher, gegebener Workflows zu gewährleisten. Die beiden anderen Ketten sind vordefiniert und decken die wichtigsten Modelle innerhalb DRIHM ab.

Das Konzept der Arbeit bietet eine Möglichkeit, um die Leistung der Workflow-Ausführungen auf jeder Backend-Ressource zu verbessern. Integriert in das Web-Portal könnte es unerfahrenen Benutzern ein besseres Verständnis der Möglichkeiten zur Verkettung von Modellen innerhalb des DRIHM-Projekts geben.

Contents

1 Introduction

Water passing rivers, landslides, floods in urban areas, destruction of infrastructure and risk of human life: people have to deal with all these threats in all regions frequently and, due to changes in the climate, such extreme weather events become also more intense. One of the biggest challenges for Hydro-Meteorology Research (HMR) is to predict such events and thus to contribute to the protection of the population.

Hydro-meteorological calculations are one of the most complex computational tasks in High Performance Computing (HPC). Small computer systems do not have any possibility to calculate such models accurately and promptly. Computing a weather event faster than real-time can save lives and beware of massive destruction. Therefore, calculations in today's research are distributed to large computer centers mostly integrated in a grid infrastructure. The present and distributed resources need to be connected efficiently. Such an infrastructure is currently being investigated in the EU project DRIHM, *Distributed Research Infrastructure for Hydro-Meteorology*. In this urgent and grid computing project, several research centers all over Europe are involved. Appliance and monitoring of the project's infrastructure is assumed by the Munich Network Management Team at the *Ludwig-Maximilians-University of Munich* (LMU) together with the *Leibniz Supercomputing Centre* (LRZ).

There are currently eight HMR models realized in DRIHM used to calculate the different factors and influences in the development of a flooding. From the clouds textures, the therefrom resulting rain, the discharge of the rivers and in the polders, up to the estimated water levels in the cities, all those factors can be simulated. The chaining of these models and the definition of the underlying basic data happens inside the DRIHM-project in a web portal, which is realized as an abstract level over the real execution layer and which provides thereby any certified user an ergonomic environment for such a submission.

As shown later in this thesis models can be chained in several ways based on two basic model layers and one ensemble generator. Creating such a workflow of HMR models is currently only possible in the portal, as shown in figure 1.1. These workflow creations are actually several single model executions which only simulate a real model chain. Of course, this consecutive execution results in a correct output, but as shown later it is inefficient and consequentially expensive.

At the back end of the portal the settings made by the user are processed in several steps. Miscellaneous preprocessing system calculations are provided on distributed clusters. For this purpose an infrastructure has been established within DRIHM which automatically creates its own environment on each cluster and ensures that all the models can access the libraries and binaries needed to run correctly. Therefore various libraries were compiled to replace wrong versioned or missing system libraries. Generally, clusters are used by many various users for many different purposes. It is therefore impossible to install all needed binaries and libraries on all these clusters, which would be necessary if they were not transferred. Although calculations within DRIHM estimate an impact on nature and population it is not the aim of any research to build various clusters with special requirements only for one project. Also economic reasons play a big role by evaluating the direction of

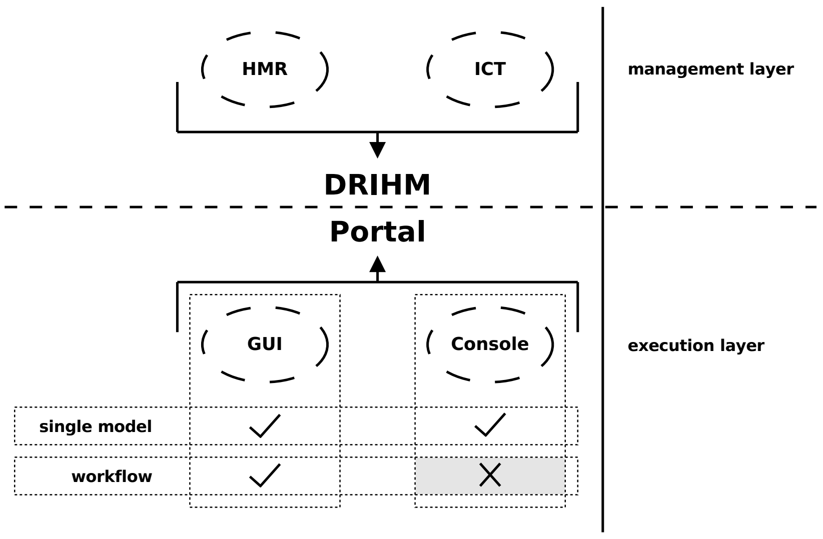


Figure 1.1: Problem statement in the back end of the DRIHM portal

the development.

But not only the clusters are different. The models within DRIHM are very sensitive with regard to the local environment, the used libraries or binaries and their versions. This needs always to be considered when setting up the various environment variables. All necessary files are transferred to the cluster before each calculation. The setup of the environment, the verification of the model call, the execution of the model and the backup of the output is monitored and executed by a bash script called *start.sh*, which forms the basis of every model run on each cluster.

Start.sh can be called not only from the portal, but also directly from command line which is especially helpful while debugging models. Therefore, the result of this thesis makes a valuable contribution and allows developers to quickly perform a complete model check and thus expose all effects to the models. Especially in a project in which many different institutions participate and design models it is important to test new files regarding all effects to other models and ensure that no interference is present throughout the entire model framework.

To design such an instrument as simple as possible pre-defined model chains are implemented into the DRIHM project within this thesis. These chains will help to determine if there are problems with new releases of models and especially in which part of the execution the failure comes up. On the other side, the pre-defined chains can be implemented into the DRIHM portal as a demonstration to new and inexperienced users showing the potential of the whole project namely to compose huge model workflows with only few clicks. To design such an appropriate realization, a modularized chain will be added to the DRIHM models, which can run every single model and almost all possible model combinations with only specifying the names of the models. Based on this modularized chain two different pre-defined chains covering the most used DRIHM models are also added.

Structure of the thesis In chapter 2 of this thesis the aim of the DRIHM project is disclosed, namely to build a bridge between HMR on the one hand and *Information and Com-*

munication Technology (ICT) on the other. In section 2.1 it is shown which HMR models exist in this project, which computational requirements need to be provided and how DRIHM overcomes the existing problems in the availability of computational power. It is also described that models can be combined in workflows but not in every sequence. All available models and their possible compositions are presented in section 2.1.3.

Section 2.2 specifies how models and workflows can be started: from the scientists view with the DRIHM front end (section 2.2.1) and from the developers view directly by using the command line on a cluster (section 2.2.2). The chapter finishes in section 2.3 with an introduction of the problem statement.

In chapter 3 a concept is developed, which integrates a workflow as a stand-alone model into the DRIHM model infrastructure and improves the actual workflow procedure. The idea and the setup of the execution environment for this enhancement is presented in section 3.1. The implementation of the so-called demoChain (*demonstration chain*) is presented in section 3.2 and detailed in section 3.2.1, where the developed modularized workflow as one DRIHM model is introduced. The main element of the demoChain, a loop for the model executions, is explained in section 3.2.2. Finally, two pre-defined demoChains are presented in section 3.3. The chapter finishes with a summary of the attained results (section 3.4).

Chapter 4 summarizes the developed concept and evaluates its performance in section 4.1. Afterwards, in section 4.2 the findings of this thesis are discussed and an outlook is given, focused on the compatibility with the whole DRIHM framework now and in future.

2 Related Work

DRIHM is an EU-funded research project to combine HMR and ICT objectives to model weather events and its influence on suburban regions. This chapter gives an understanding of the project in section 2.1, especially of the requirements of the hydro-meteorological science and its embedment in a supercomputing infrastructure. The integration of HMR computations in present grid resources is detailed in section 2.1.2. All HMR models, which are included into the DRIHM project, are outlined in section 2.1.3

In section 2.2 it is explained how these models and also complete workflows can be configured and executed in the DRIHM front end (section 2.2.1) and in the back end via command line (section 2.2.2). Section 2.2.3 provides an insight into the used grid environment.

It will be pointed out that a composition of models as one workflow is only possible in the DRIHM portal and not within the back end, and so present the problem statement of this thesis in detail in section 2.3, showing particularly that the chaining in the portal is inefficient and expensive with regard to the computing time and transferred data.

2.1 DRIHM project

The DRIHM project faces up to the major problems in HMR: composing different models to workflows and the availability of HPC systems [DCG$^+$14]. Bringing both together could push the whole scientific branch to a higher level. Gaining more computing power opens new possibilities to current and to new scientist using HPC systems. Before DRIHM only a few HMR scientist were able to compose different HMR models on a powerful resource. It was and is a big barrier for many institutes to run huge and complex simulations.

Of course, in the last years the number of HPC systems all over the world increased very fast. But with this development also the data volume increased rapidly, since HMR models usually handle a big amount of data. More computing possibilities always invite scientists to increase the detailing in persistent models. Particularly in HMR, where big landscapes are modeled, the grid spacing of the domains can always be reduced. And this trend will increase in the next years.

The newly created supercomputing centers will reach their upper load limit at a particular time. At this point, the DRIHM project starts to build a bridge between existing grid computing systems and the desire for more computational power on the side of the scientists across all HMR disciplines. The project wants to set new standards of HMR in Europe and possibly worldwide by establishing a new collaboration framework between HMR and ICT scientists and by making a big step beyond the state of the art in composing weather forecasting models [BPQ$^+$12].

2.1.1 Hydro-Meteorological Research (HMR)

The hydro-meteorological science has made improvements in modeling over the last years, e.g. new models to collect and analyze data were released. But the huge quantity and the

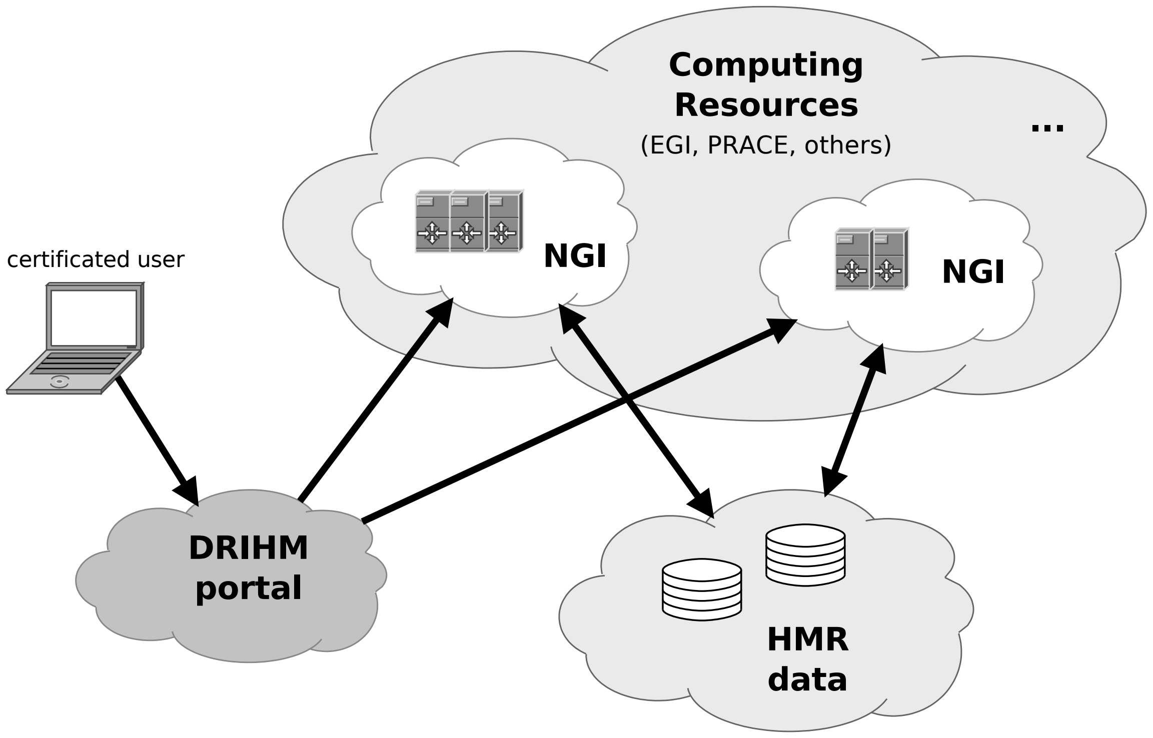


Figure 2.1: The modeled DRIHM infrastructure, based on [DRI15d]

complexity of datasets are stumbling blocks and change the handling of data. "As a matter of fact, observational data, HM models and ICT [..] resources are not generally available at the same time and are often distributed in an uneven manner between different research institutes, HM services and operational agencies" [BPQ+12].

But the data and the resources must be available in real-time to make reliable predictions on weather events. It does not make sense to observe data if no simulation of a model is possible and it is not economic to maintain a huge computing cluster if there are no datasets to compute. Data sets must be available which means they must be easy to locate and the needed permissions for use must be easy to obtain. It is also important to have a computing environment which can handle different data types and an infrastructure which supports HMR scientists during the computation. All this will be most important if the computation is a realtime simulation and the results are supposed to help to prevent damage of urban areas or even of human life in urgent cases [BPQ+12].

It is a really needed aim to deploy a simple and ergonomic interface for scientists to compose their data sets and models while not taking care of any underlying e-infrastructure and specific implementations of the HPC system. The DRIHM project is to close this gap between the two departments HMR and ICT [BPQ+12].

2.1.2 Integrating HMR resources in grid resources and HPC

The DRIHM computing environment bases on a European grid computing infrastructure in which several *National Grid Initiatives* (NGIs) collaborate. The goal of these NGIs is to

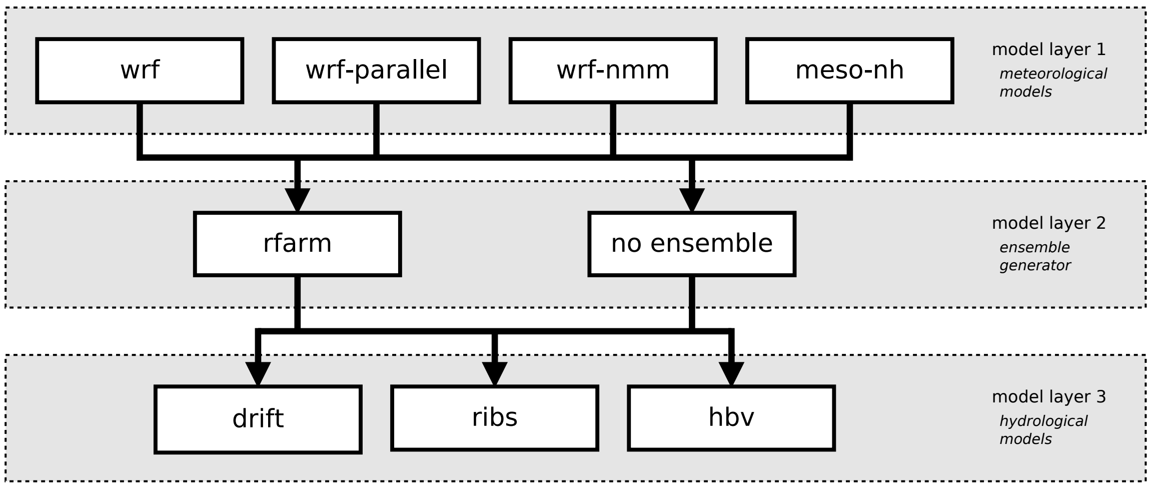


Figure 2.2: HMR models: overview and possible chaining

provide a reliable and secure e-infrastructure in the European countries and to establish a Europe-wide grid and cloud infrastructure for scientific use [ND15]. A simplified view on this infrastructure can be seen in figure 2.1 which shows the connection between a certified user using the DRIHM front end and the computing resources with the needed HMR data on other servers. A detailed insight into the distributed infrastructure will be provided in figure 2.4 on page 9.

With the development of a user-friendly interface DRIHM aims to abstract provided HMR services from the specified e-infrastructure. The implemented abstract level encourages multidisciplinary and international collaboration between meteorologists, hydrologists and other potential scientists from other scientific areas. This HMR e-Science environment enables unknown possibilities in the composing of different HMR models, processing tools and data. Through this abstract level DRIHM will allow specialists to enter the e-Science environments much more easily [DRI15d].

With the decision of not building up an own environment the responsibility for the reliability and maintenance of the used clusters lies with the operating company. It is more cost-effective and robust than self-managed systems. Using external clusters only when required is of course an optimum solution from an HMR point of view.

2.1.3 Hydro-Meteorological models

The DRIHM project offers with the presented infrastructure several HMR models which can be computed alone or in a workflow where up to three models can be composed. The possible workflows can compare up to one meteorological model with one or none ensemble generator and one hydrological model. A categorization in these three layers is shown in figure 2.2.

Meteorological models The meteorological models which are included into the DRIHM framework are WRF-NMM, WRF-ARW, Meso-NH and RainFARM. All models run at kilometric scale resolutions up to several hundred kilometers (mesoscale). The *Weather Research and Forecasting* (WRF) model is a modern mesoscale numerical weather prediction system.

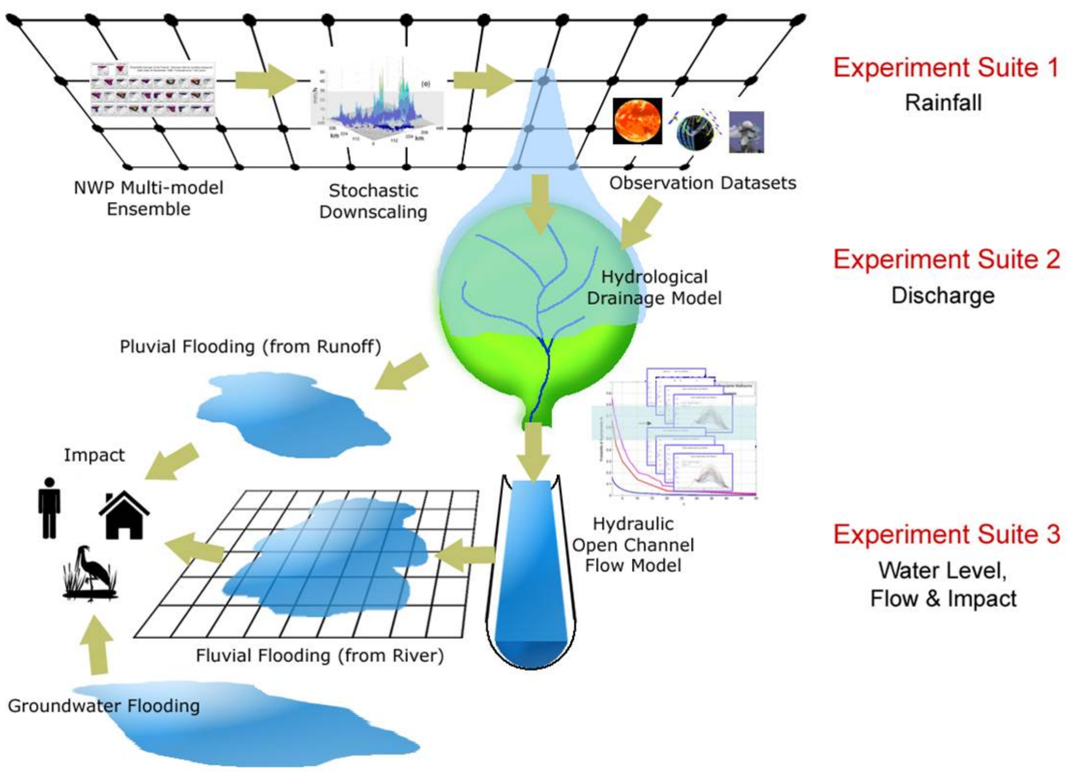


Figure 2.3: Schematic diagram of the forecast chain for floods [DRI15e]

Two versions of the model exist differing in the description of their dynamical cores and in the underlying grid schemes for the computations. Besides a sequential WRF model the two main models are on the one hand the *Advanced Research WRF* (ARW, within DRIHM called wrf-parallel) and on the other hand the *Nonhydrostatic Mesoscale Model* (NMM, included as wrf-nmm). These models were and are developed by several US research centers [HCG+15].

The *mesoscale non-hydrostatic model* (Meso-NH), the fourth model in layer 1, is also a non-hydrostatic mesoscale atmospheric model developed by a French laboratory. Another planned French model, AROME, is not included in DRIHM in the current state. The *Rainfall Filtered AutoRegressive Model* (RainFARM, in DRIHM rfarm) is a method for the realization of stochastic rainfall downscaling. It can be easily applied to the rainfall forecasts provided by the other mentioned meteorological models before as an ensemble generator [HCG+15].

Hydrological models Three different hydrological models are implemented in the DRIHM project: DRiFt, RIBS and HBV. DRiFt (*Discharge River Forecast*) is a graded rainfall outflow model based on a geomorphologic approach. The *Hydrologiska Byråns Vattenbalansavdelning* (HBV) model is also a semi-distributed hydrological model developed by a Swedish institute. RIBS (*Real-time Interactive Basin Simulator*) is a physically based distributed model which computes hydrologic basin responses to spatially distributed rainfall inputs [HCG+15].

In figure 2.3 a complete, schematic diagram of a forecast chain is presented. The so-called rainfall layer (cf. figure 2.2 layer 1 and 2) contains the combination of the different meteo-

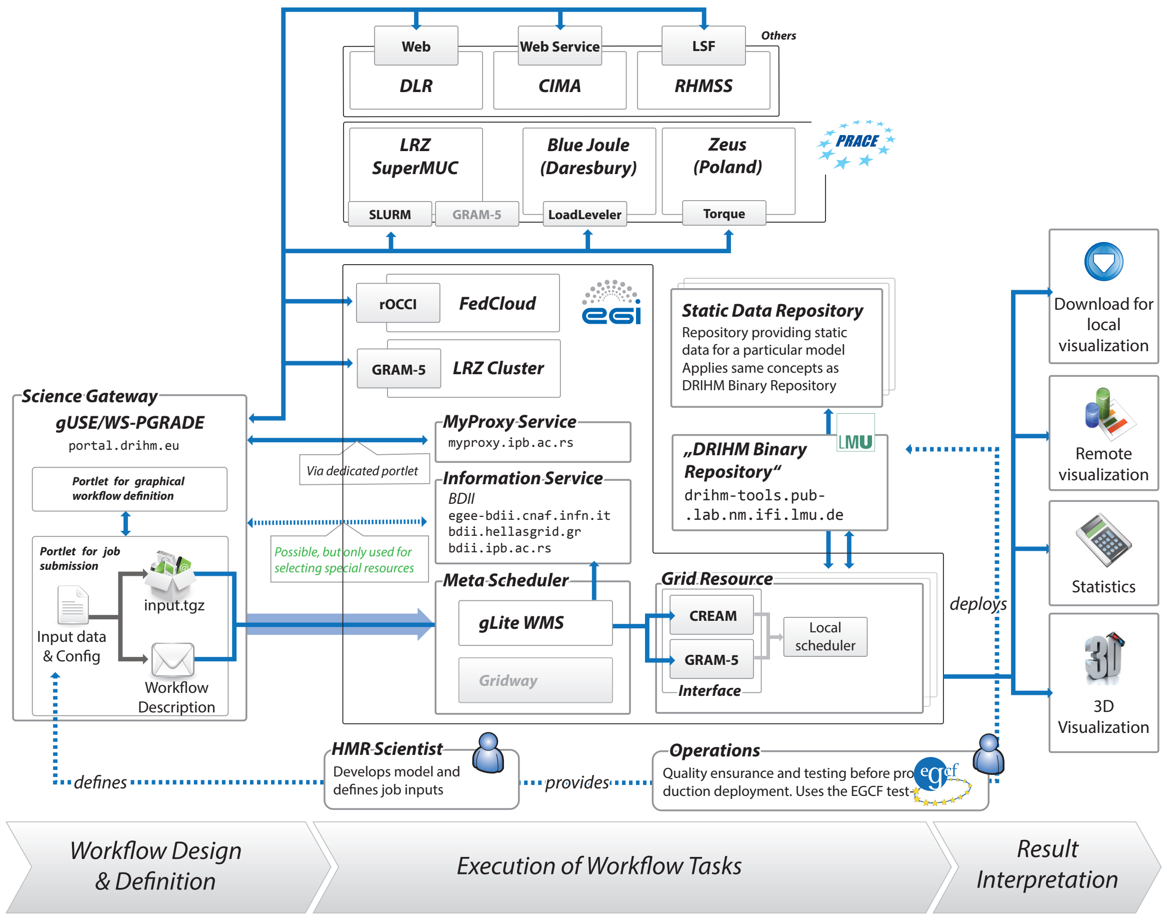


Figure 2.4: DRIHM Distributed Computing Infrastructure [DRI15b]

rological models and the ensemble generator as shown. The discharge layer (cf. figure 2.2 layer 3) concerns the fusion of rainfall predictions (from the rainfall layer) with corresponding observations, which are needed as an input for multiple hydrological models to enable the before described production of river discharge predictions [DRI15e].

Hydraulic models The water level, flow and impact layer addresses the execution of hydraulic model compositions in different modes. These models can estimate river stage, discharge and impact generated by a flood event. Within DRIHM the models Mascaret/RFSM and Delft3D are designated to fulfill this requirement, but they are not part of the mentioned e-infrastructure at the moment and are not considered in this writing.

2.2 Executing models in DRIHM

The DRIHM project introduces an abstract layer for the scientific users, who set up and run their models in a web portal. The real computation is done in the back end of DRIHM on the distributed grid resources all over Europe with a bash script. Figure 2.4 portraits the *DRIHM Distributed Computing Infrastructure* (DDCI) to control and execute these models.

This extensive figure shows the complete process starting with the design of a model run in the 'Science Gateway', the DRIHM front end and followed by the tethered resources all over Europe. They are accessed via PRACE (*Partnership for Advanced Computing in Europe*) Research Infrastructure, EGI (*European Grid Infrastructure*) and others. The figure also shows the authentications barriers to grant access to submit a job to the grid resources. The connected 'DRIHM Binary Repository' is a LMU managed server which provides all needed software. Finally, the scientist who does not gain insight to this process on the back end, can access the results of his job on the web portal.

In figure 2.4 all processes from the scientist's action in the front end to the real execution in the back end can be followed. This distributed computing infrastructure is the main achievement of the DRIHM project, a middleware which makes any scientist able to run any kind of complex HMR models within minutes. Within the front end web portal the certificated user can simply choose from the mentioned HMR models and do all configurations with only a few clicks. Models can also be combined to a workflow in this place.

The next sections present the scientist view on the front end and afterwards the execution on the back end.

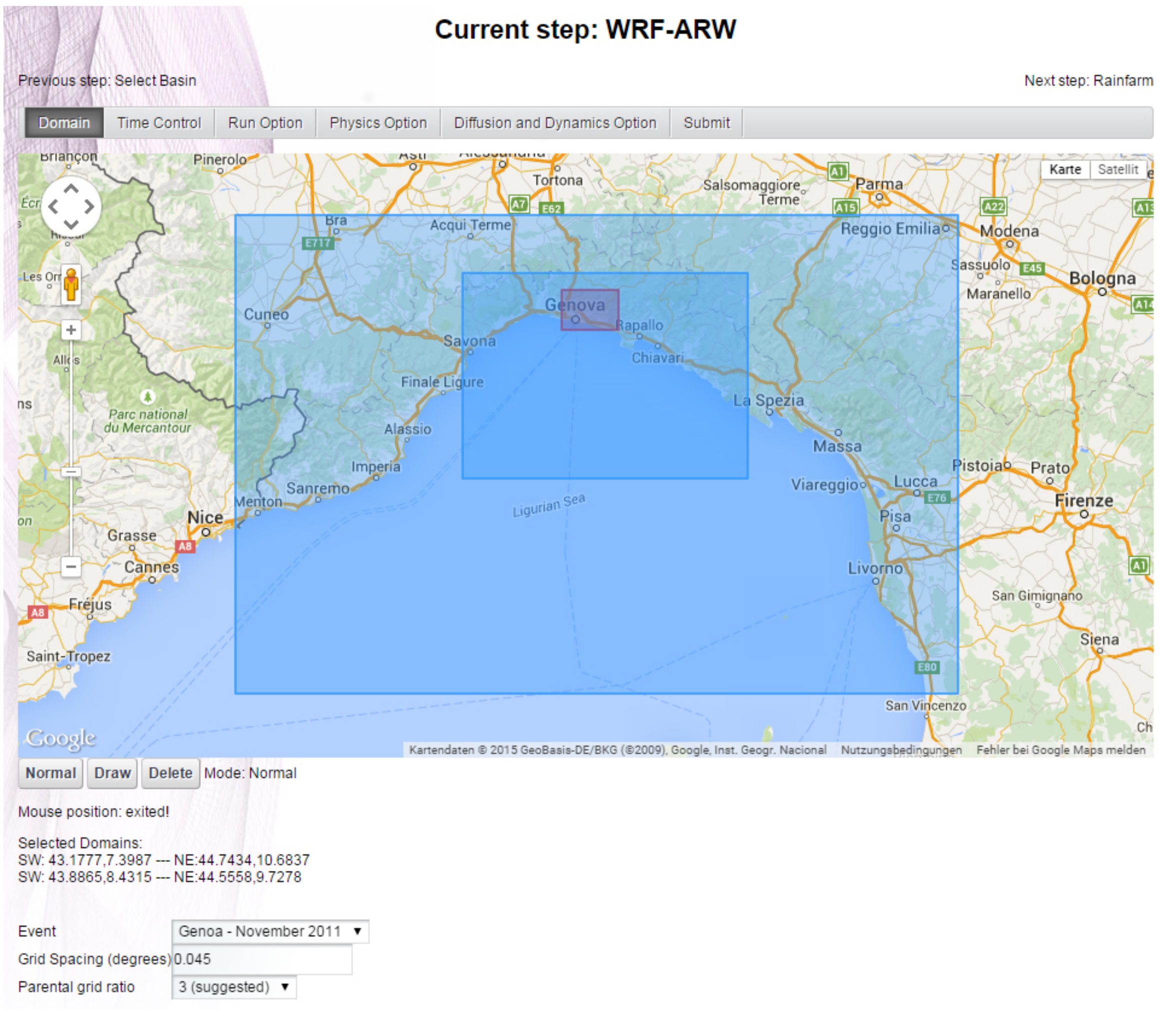
2.2.1 DRIHM front end: the portal

The developed front end presents the user several tools to adjust and run HMR models. A model depended portlet covers all possible settings and parameter definitions. In figure 2.5 a tab of the WRF-ARW configuration is shown in which the selection of an appropriate basin and the definition of up to three domains locate the area in which the model calculates its simulations. These are together with the resolution of the computation, the grid spacing level, the most important set screws for this model. For all models of layer 1 (cf. figure 2.2) also time steps of the calculation points and more important the physic, diffusion and dynamic options of each run need to be defined. With these settings the input data for the model is measured from a preprocessing system like WPS (*WRF Preprocessing System*).

In the portal the input for models from layer 2 and 3 needs to be selected out of a list if no model from layer 1 is included in the workflow. Theses input files are pre-calculated on a special case, the 2011 flood in Genoa. A main part of the DRIHM calculations are based on this case because it is very good documented and included into the DRIHM database.

All settings to make a model execute are made in the web portal, an abstract layer front end. The user clicks are translated into several setting files which are added to the model run. These setting files are extensively depending on the model, but the DRIHM portal reduces a real huge amount of coding time for any scientist to a more user-friendly, easy and quick configuration in a browser. The portal back end servers also create all other needed files to run the model in the back end of the project like additional information files on the basin.

Not only the starting and configuring of a model can be done in the portal also the monitoring is an important function included in DRIHM. The status of any own model run and the results of the calculations are presented in a monitoring page where every run, single model or workflow, can be traced. An example is shown in figure 2.6. Also the standard output and error output from the cluster, which the model was computed on, are integrated there. The user gains insight what steps of the model ran successfully or also where an error occured. Therefore the portal includes a feedback function a scientist can react on and adapt the configurations of the run if the result is not satisfactory.



Current step: WRF-ARW

Previous step: Select Basin

Next step: Rainfarm

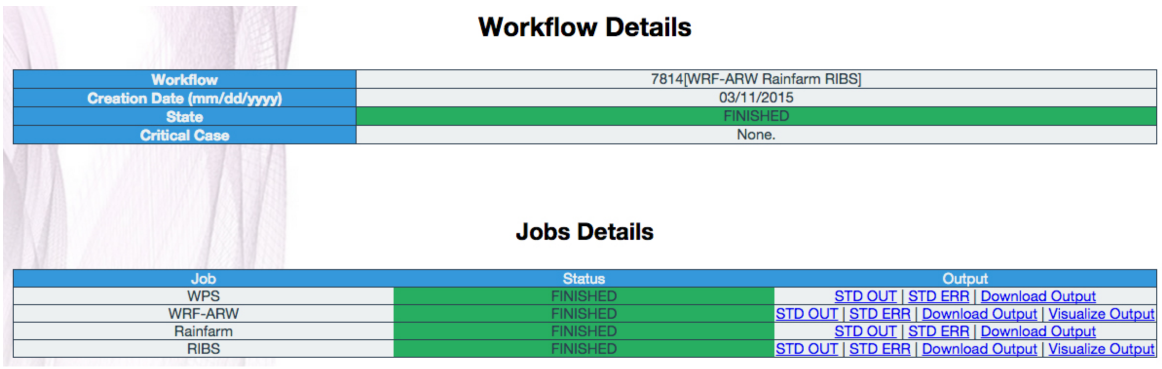
Mode: Normal

Mouse position: exited!

Selected Domains:
SW: 43.1777,7.3987 --- NE:44.7434,10.6837
SW: 43.8865,8.4315 --- NE:44.5558,9.7278

Event	Genoa - November 2011
Grid Spacing (degrees)	0.045
Parental grid ratio	3 (suggested)

Figure 2.5: Screenshot from the DRIHM portal web front end configuring WRF-ARW [DRI15c]



Workflow Details

Workflow	7814[WRF-ARW Rainfarm RIBS]
Creation Date (mm/dd/yyyy)	03/11/2015
State	FINISHED
Critical Case	None.

Jobs Details

Job	Status	Output
WPS	FINISHED	STD OUT | STD ERR | Download Output
WRF-ARW	FINISHED	STD OUT | STD ERR | Download Output | Visualize Output
Rainfarm	FINISHED	STD OUT | STD ERR | Download Output
RIBS	FINISHED	STD OUT | STD ERR | Download Output | Visualize Output

Figure 2.6: Screenshot from the DRIHM portal web front end monitoring a workflow [DRI15c]

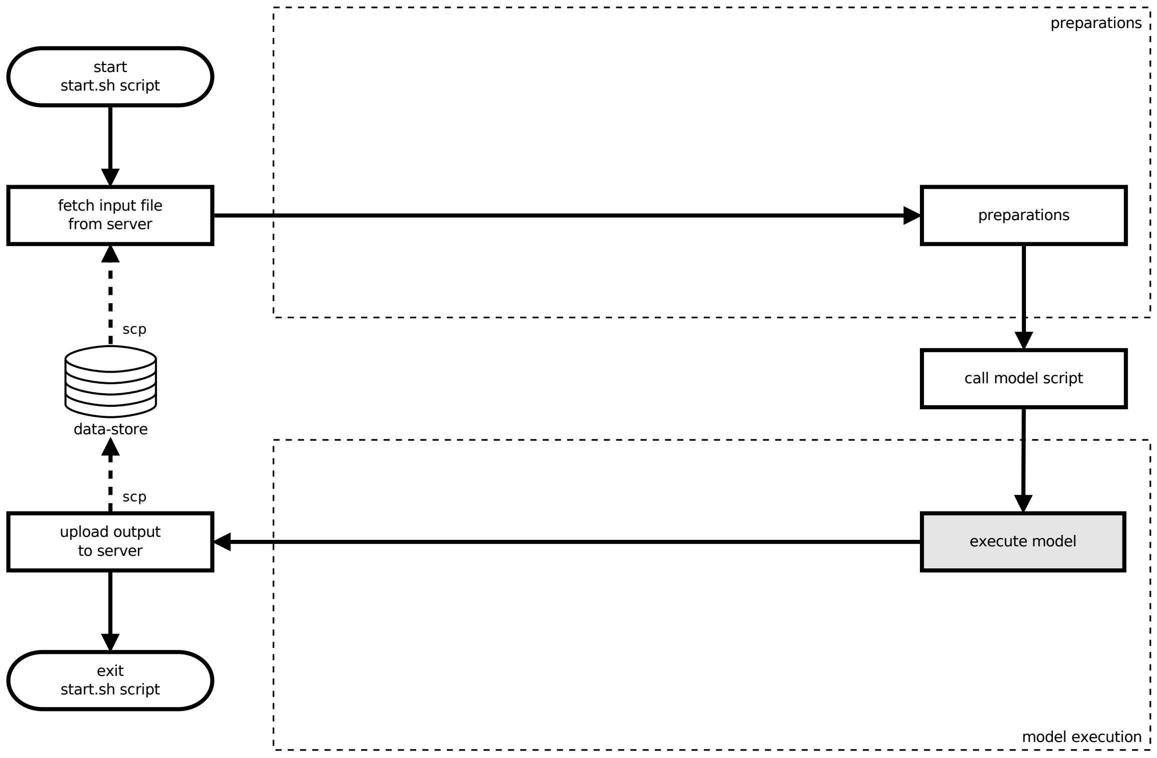


Figure 2.7: Chart of the start.sh script and its main operations

The real execution environment is covered and the user has no insight into the computation of the pre-processing systems, the model run on the clusters and the output management. Due to the implemented grid environment also the location of the included cluster is not selectable and not even visible. The user is also not able to know that the models in a workflow he designed in the portal are not calculated in one execution on one cluster. They are all separate and single executions on potentially different clusters. Already at this point the main problem crystallizes namely that workflows are not handled as workflows but as different single model runs. This will be detailed in section 2.3.

2.2.2 DRIHM back end: the bash script start.sh

Starting a simulation of HMR models in the DRIHM portal is as shown an abstract execution. On the portal's back end several processes are started which can also be started directly on the console. At the end, every job submission from the portal results in a very simple command line execution on the destinated resource. The main part of the execution is a bash script file, the so-called *start.sh*. It follows a strict syntax which also could not be changed for this thesis:

./start.sh <model> <jobid> <input> <output>. Besides the name of the model a job id number needs to be specified as well as an input and an output filename. With this information every model can be started on any resource.

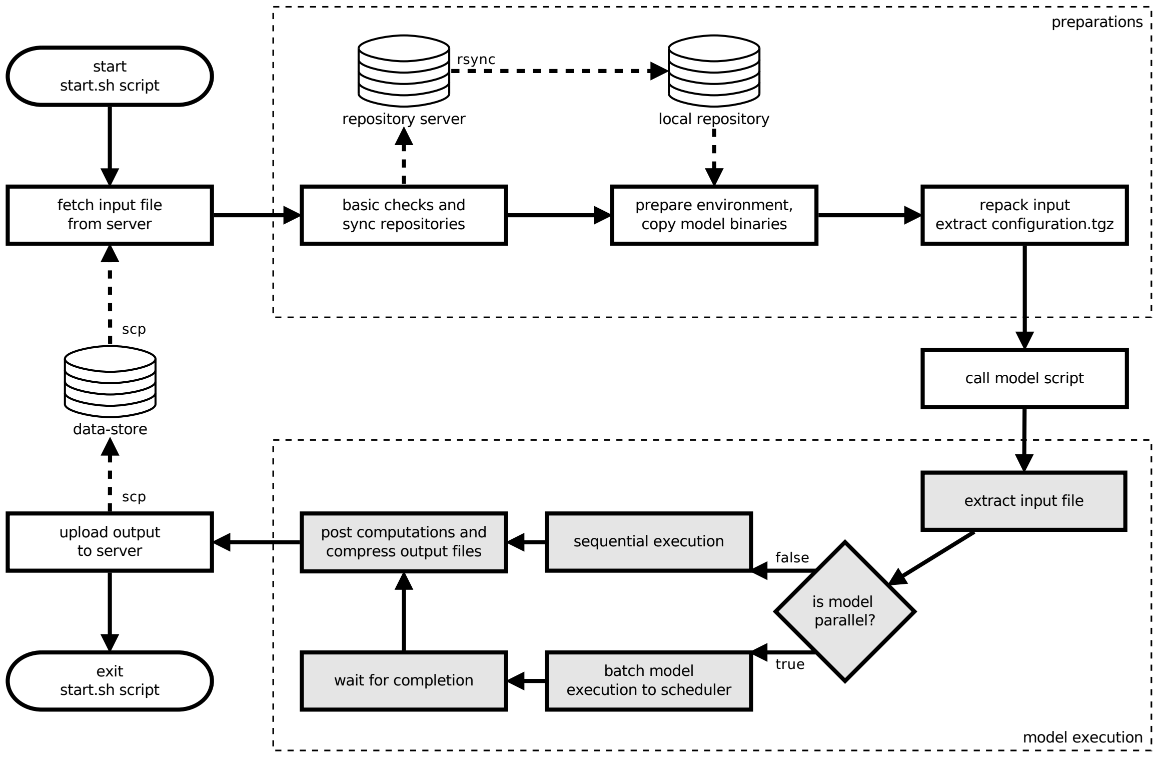


Figure 2.8: Chart of the start.sh script and its operations in detail, based on figure 2.7

The start.sh script All needed checks and definitions to run a model are made in this script. The chart in figure 2.7 shows the main operations made by start.sh. For a complete look at all steps of the execution plot the chart in figure 2.8 includes all intermediate steps.

The script will be transferred to any cluster in the e-infrastructure with the addition of some elementary files. This includes keys for the synchronization of the model's binaries as well as libraries or precompiled system binaries. They are essential because as mentioned before HMR models a very sensitive and necessarily need a precise defined system environment. Due to the fact that not all clusters have a comparable set up (e.g. different operating system, different system libraries, modules or binaries) several gigabytes are moved to the relevant cluster. If the cluster does continuous computations the transferred data is possibly to stay on its storage system and is only synchronized on each call of start.sh to stay up-to-date.

Preparations Before a model can start its actual computation the transferred system data must be integrated, paths and local variables must be set. With the call of start.sh also several parameters are assigned as shown. The most important parameter besides the model is the name of the input file, which needs to be fetched from the data-store, a back end server in the DRIHM e-infrastructure. Afterwards, if it is available there at all, the script checks the syntax and the local environment for all needed parameters. Anywhere an error occurs the following exit is trapped by a tidy function which removes all transferred data but start.sh, as well as the temporal job directory of the model. To avoid any changes on the local repository, which contains libraries, model and other binaries as well as the actual model scripts, every model computes in this temporal directory, the so-called *jobdir*. The model binaries are copied there and could also be changed by a model execution without any influence on other runs because the repository itself stays untouched.

It is possible to avoid all this data to be removed in case of debugging by setting a parameter in start.sh. After all checks are made and the local repository is updated the preparation of the execution environment starts. This includes to set all path variables, like the *ld library path* to make the libraries and binaries available the model script might need.

Each model needs configuration files specifying the input. The start.sh script standardizes this and forces all configuration files to be included in an archive named *configuration.tgz* and to be located in the same directory as start.sh. From there it is extracted into the jobdir of the model. Because several models can be combined in several ways the output files from one model and the expected input files for the next differ usually in the needed filename. Therefore a 'mapping.txt' file can be added to the configuration files containing a mapping route of the input file which effects a repack of the input archive. The input itself is unpacked from the actual model script which is called from start.sh afterwards.

Model execution The model scripts differ in the execution layout, particularly if they have a parallel part. In figure 2.8 the shown grayed part is executed by the model script. The meteorological models wrf-parallel, wrf-nmm and meso-nh compute in parallel and might only have a sequential part at the end of the script. All others are completely sequential. HMR models like wrf-parallel tend to be very big simulations. That is why they need a very fast connected and well performing platform for their computations. Therefore, as shown in section 2.1.2, European grid resources are integrated in DRIHM. For one model run at least 32 physical cores are needed and the computation can still run up to 48 hours and longer if the detailing is increased. To run more models at once, several nodes need to be available. The detailed execution environment is presented in section 2.2.3 as well as the job scheduler, which starts and controls the parallel jobs on the used clusters.

Completing the computation of a model, the generated output is compressed to an archive, which is transferred to and stored in the data-store. After this transfer the start.sh finishes with the already mentioned tidy function.

2.2.3 Grid environment

The grid resources are granted by EGI and are located in Croatia, Germany, Greece, Italy, Macedonia, the Netherlands, Serbia, Spain and Poland [EGI15]. A powerful and the most used resource in this thesis is the Linux-Cluster provided by the *Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities* (LRZ) in Germany, running with the Globus Toolkit 5, which includes among other things software for data and job management or security [VA12]. The other resources are accessed with gLite (*Lightweight middleware for Grid Computing*), another middleware [DCG$^+$14].

To access grid resources certificates are needed. Without them any try to send a job to a resource is refused. All these clusters are included in a well performing grid environment which provides a fast and scalable possibility to use present resources reasonable and efficient. The LRZ cluster for example consists of several segments with different types of interconnects and different sizes of shared memory. The DRIHM used segment is a MPP cluster with 16-way AMD-based nodes and Infiniband interconnect [LRZ15]. The Cluster runs its jobs in parallel using mpi and the job scheduler SLURM.

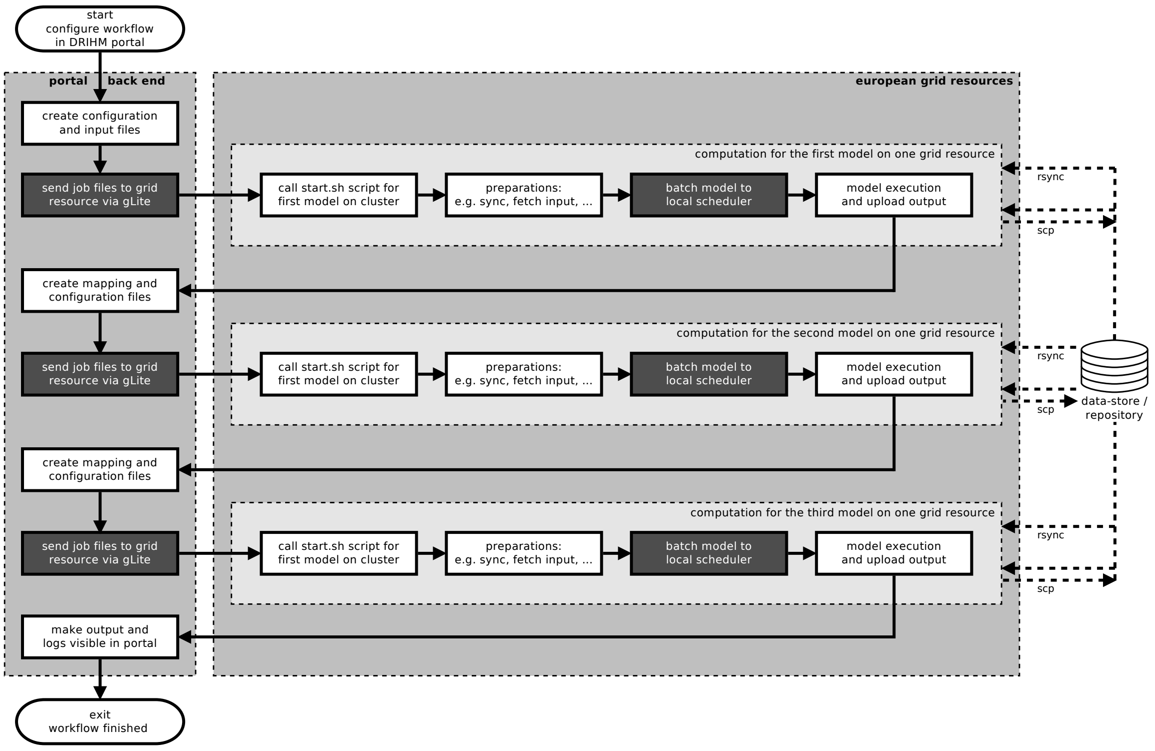

start
configure workflow
in DRIHM portal
portal
back end
european grid resources
create configuration and input files
send job files to grid resource via gLite
computation for the first model on one grid resource
call start.sh script for first model on cluster
preparations: e.g. sync, fetch input, ...
batch model to local scheduler
model execution and upload output
rsync
scp
create mapping and configuration files
computation for the second model on one grid resource
data-store / repository
computation for the third model on one grid resource
make output and logs visible in portal
exit
workflow finished

Figure 2.9: Chart of the execution of a portal-defined workflow on grid resources

For large-scale parallel programming the *message passing interface* (mpi) is the most heavily used paradigm at the moment and is certainly used at the LRZ and in all DRIHM models. To control the parallel jobs the LRZ and other clusters use SLURM (*Simple Linux Utility for Resource Management*), an open-source manager designed for clusters of any scale. The main functions are to allocate access to resources (like computer nodes), provide a framework for managing a parallel job on a set of the allocated nodes and to queue pending jobs within a cluster. Today it provides its workload management on many of the most powerful clusters worldwide [Sch15]. To batch jobs to SLURM it was integrated into start.sh and is at the moment of writing the only supported scheduler within DRIHM.

2.3 Problem conclusion

The complete workflow as shown in section 2.2.2, precisely in figure 2.8, is to be repeated each time a model run is called. If a user defines a workflow in the portal with the maximum of three models, the whole start.sh procedure is executed three times. A chart of this execution is shown in figure 2.9. This is obviously very ineffective because especially all preparation steps are iterated.

Furthermore, in a distributed grid infrastructure all files must be transferred to each cluster. For this reason such an execution produces a huge amount of traffic. In figure 2.9 it is marked with *rsync* for the transfer of the DRIHM repositories and *scp* for the fetch of the input and the put of the output file to the data-store. Of course, if a cluster is used frequently, a local storage is provided which only needs to be synchronized every time a

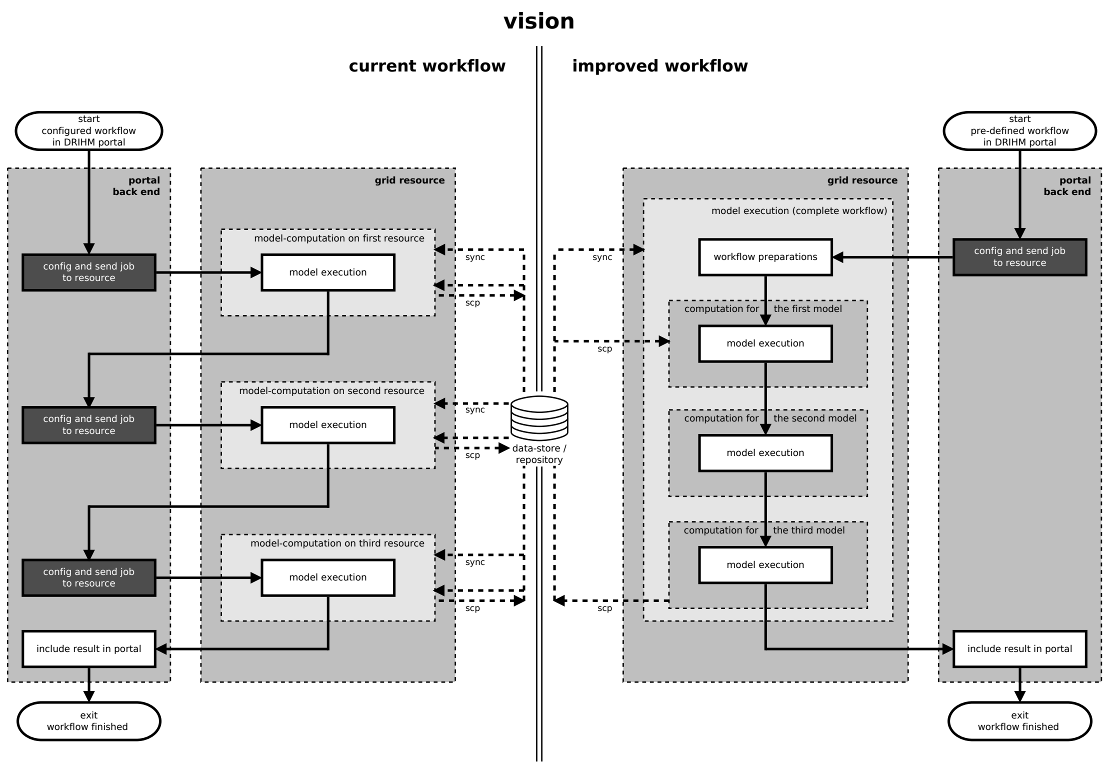

vision
current workflow
improved workflow
start configured workflow in DRIHM portal
portal back end
grid resource
model-computation on first resource
config and send job to resource
model execution
sync
scp
model-computation on second resource
model-computation on third resource
data-store / repository
include result in portal
exit workflow finished
start pre-defined workflow in DRIHM portal
model execution (complete workflow)
workflow preparations
computation for the first model
computation for the second model
computation for the third model

Figure 2.10: Comparison between a portal and a resource managed workflow

model runs. But the data transfer is likely to last long and meanwhile blocks all allocated nodes on the cluster. In times of increasing electricity rates the costs are one limiting factor of executing science models on HPC clusters. Therefore, it is essential to be as efficient as possible.

Submitting a job to a grid resource is controlled by a scheduler. So every workflow like shown in figure 2.9 is scheduled three times only to access a cluster. Thereby another problem comes up with the authentication of the certificates of the user who started the workflow in the portal. To access any grid resource a complete authentication procedure is necessary. The obligatory two certificates, one for the DRIHM project and one for the European grid resources at all, will be checked for every single model run via gLite, in the workflow in figure 2.9 also three times. Depending on the registration authority grid certificates need to be renewed approximately every 24 hours due to security reasons. A submitted workflow in the portal can so fail if the first computation takes too long and for the following models the certificates are not renewed in the portal. This problem cannot occur if a workflow is submitted in only one transmission.

A workflow composed in the DRIHM portal is also not efficient because every single model runs as a single job. And because all clusters are managed by a job scheduler the model executions are sorted in potentially huge queues with waiting times up to several hours or days each time. If no parts of the cluster are reserved for DRIHM executions, which would certainly effect higher costs, real-time simulations are often not possible without any prioritization. Especially in a peak time a workflow would have to wait three times in such a scheduler. This is not efficient and, if a result is needed urgently, of course unacceptable.

In the end, also debugging model updates in such an unreserved cluster environment is not feasible. Trying to find a possible error somewhere while always waiting up to hours till the error occurs costs certainly man and computing hours and needs to be improved.

Vision Comparing these problems, a vision of an improvement is designed in figure 2.10, which leaves most of the mentioned disadvantages behind. The figure illustrates a the state of the art workflow in comparison with the idea of an enhanced workflow transmission. On the left side in this figure the current workflow submission is shown, which is managed by the DRIHM portal.

It becomes apparent that for every model in the workflow a separate transmission including separate data transfers is needed. This is done by design of the DRIHM project, which is constructed to include also models which cannot be executed on every included cluster within the DDCI, but need a special execution environment, e.g. some impact models (hydraulic models) compute on windows based servers.

However, taking as a basis, that all models can be computed on the same resources and in the same basic environment, an improvement of the current workflow is obviously possible. This can be assumed for all possible model chains shown in figure 2.2 on page 7.

To reduce the current job transmissions and data transfers another kind of model composition will be developed: a complete workflow in only one transmission as shown on the right in figure 2.10. At first glance the improvement of the workflow is visible. The needed synchronization calls and data transfers are minimized and the job submission scheduler is entered only once. This adaption would improve the performance of every HMR workflow within DRIHM in its described state.

3 Efficient chaining of models in DRIHM

This chapter will present the concept and the implementation of a real workflow in the back end, which avoids the mentioned problems before in section 2.3, namely unnecessary data transfers and avoidable queueing times. The concept of this workflow is presented in section 3.1, which summarizes the advantages and exposes, that the complete workflow as a whole is handled like a single model. Therefore, the so-called *demonstration chain* (demoChain) is implemented like any other model within DRIHM and henced controlled by the start.sh script.

The needed preparations to include the demoChain as one model are shown in section 3.1.1. It will be pointed out that this is similar to the preparations shown before for any other single model. The resulting back end workflow will be implemented as a modularized chain, which can contain all possible model chains within DRIHM. All preparations for this purpose, e.g. a model chain check or configuration file arrangements, are detailed in section 3.2.

To enable all possible model combinations and to make any chaining of models more efficient a loop containing only essential functions replaces the current model execution and is shown in section 3.2.2. The specifications of the modularized demoChain, which is added to the DRIHM framework as an executable model, are presented in section 3.2.3. Based on this modularized workflow two pre-defined chains are also added as static models into DRIHM. They cover the most important HMR models and are introduced in section 3.3. The chapter concludes in section 3.4 with a summary on the developed demoChains.

3.1 Concept

In section 2.3 the problems of the state of the art DRIHM workflow were presented in detail. The main fact is the non-existence of a complete workflow queue in the back end. Therefore, all workflows are computed in single runs, one for every included model (c.f. figure 2.9 on page 15), and it is impossible to use the existing infrastructure at an optimum. Of course, this is done by design of the DRIHM project, but for comparable models it is consumptive of time, because of avoidable traffic and transmissions. This will be optimized with the addition of the demonstration chains to DRIHM.

At the moment of writing, all models within the DRIHM project can be executed on the same clusters and in the same environment. So any model chain can be improved, because all models have the same basis. To reduce the amount of traffic every workflow, which consists of up to three different models (according to figure 2.1.3 on page 7), is combined into one model as shown in figure 3.1. This leads to a significant reduction of traffic because the mentioned repositories need to be transferred or synchronized only once instead of three times. Also intermediate results will be managed on the resource locally and will not be transferred to the data-store and fetched from there again for the next model, which was impossible while running a workflow in single executions.

The second main effort is the reduction of the waiting time at the scheduler for accessing a grid resource. If transmitting only one model, which includes the whole model chain,

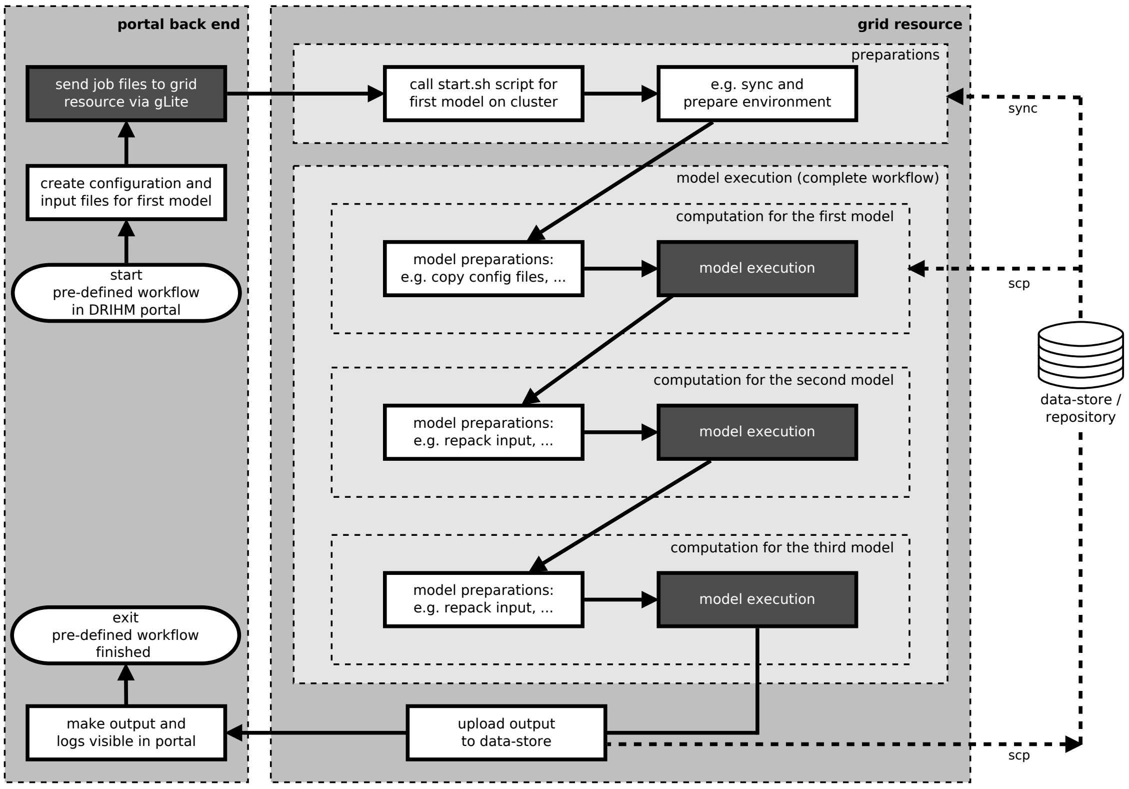

portal back end
grid resource
preparations
send job files to grid resource via gLite
call start.sh script for first model on cluster
e.g. sync and prepare environment
sync
create configuration and input files for first model
model execution (complete workflow)
computation for the first model
start pre-defined workflow in DRIHM portal
model preparations: e.g. copy config files, ...
model execution
scp
computation for the second model
data-store / repository
model preparations: e.g. repack input, ...
model execution
computation for the third model
model preparations: e.g. repack input, ...
model execution
exit pre-defined workflow finished
make output and logs visible in portal
upload output to data-store
scp

Figure 3.1: Chart of the execution of a pre-defined workflow as one model, compare to figure 2.9

the workflow will be queued only once and therefore reduces the waiting time and the authentication barrier detailed in section 2.3.

3.1.1 Preparations

The model chain developed in this thesis is likely to be a model script itself. It is processed by the start.sh script like any other model script. The preparations as shown in figure 2.8 on page 13 are exactly the same. The demoChain also creates a temporal job directory in which the single models within the chain have their own job directory. The data structure of the working directory is very similar, just with one added layer to combine all models' directories. Also the mentioned tidy function, which cleans up all data, only needs to remove the temporal directory of the demoChain which includes all other files.

As shown in figure 3.2 the structure of the complete demoChain is based on the execution of any other model shown in figure 2.7 before. The start.sh script prepares the same execution environment like for any other model. As detailed in chapter 2 before the main problem, which makes the model chaining inefficient, is that all models are run in a separate call of the start.sh script, in which many steps need to be run several times. To avoid this the main idea behind the developed model chaining in this thesis is to run all operations and function only one time. The file synchronization within the preparations section is one of the most traffic and cost consuming part. This is eliminated due to the fact that all model executions of one workflow and not only one single model are based on one synchronization call as shown in figure 3.2. This is the greatest benefit of this concept.

But also this call of the start.sh script needs to fulfill the syntax requirements which demand e.g. a given input file when calling the script, which will be fetched in the preparations section. The fetching of this input file from the data-store server is done as one of the first steps even before synchronizing the local repository. This is specified in the start.sh script and could not be changed for this concept. But before the script knows which models within the demoChain will be executed, which is read from a file afterwards, the input should be fetched from the data-store. This is not possible. Therefore a blank input file was placed in the data-store which should be specified in every demoChain call as the input filename. Every user should be aware of that. Of course, every other input can also be fetch without correlating the correctness of the start.sh script. But this input is as mentioned discharged and to save traffic and time it is not recommendable to use another input because the size of these files can increase to several gigabyte depending on the appropriate model.

3.2 Implementation

After these preparations are done the already known model execution starts by calling the appropriate model script. In case of the demoChain it is a bash script which includes all needed functions to run a complete workflow. Before the first model can start some additional preparations need to be done.

3.2.1 Workflow as one model

The main idea behind implementing this demoChain is to design it modularized. This means that one script can catch all possible workflow-calls without changing the script itself. For this reason a loop was integrated which repeats all necessary steps, but only the elementary ones. The reduced amount of function calls maximizes the advantages regarding the calculating time and transferred data. But to prepare the environment for such an execution some other steps need to done ahead.

Workflow preparations Because the demoChain covers all possible model compositions also configuration files must be available for any combination. This means that for all possible workflows and parts of them separate input files, configurations files and mapping files needed to be pre-configured. Because these configuration files are very small, they are usually only text files, it is not recommendable to fetch them separately. This would again produce more traffic and obviously take more time to transfer them. Therefore they are all fetched within only one tared and gzipped archive and they are extracted into the job directory of the demoChain. From there the required one will be copied into the root directory from start.sh within the model execution loop and renamed to *configuration.tgz*, where it is expected.

Delivered workflow revision As mentioned in section 2.1.3 not all models can be chained in any order or at all. Hence a formal logic needs to be implemented to make sure that the given models are on the one hand combinable and on the other hand exist at all. The models are specified in an extra file named models.do, which needs to be readable and located in

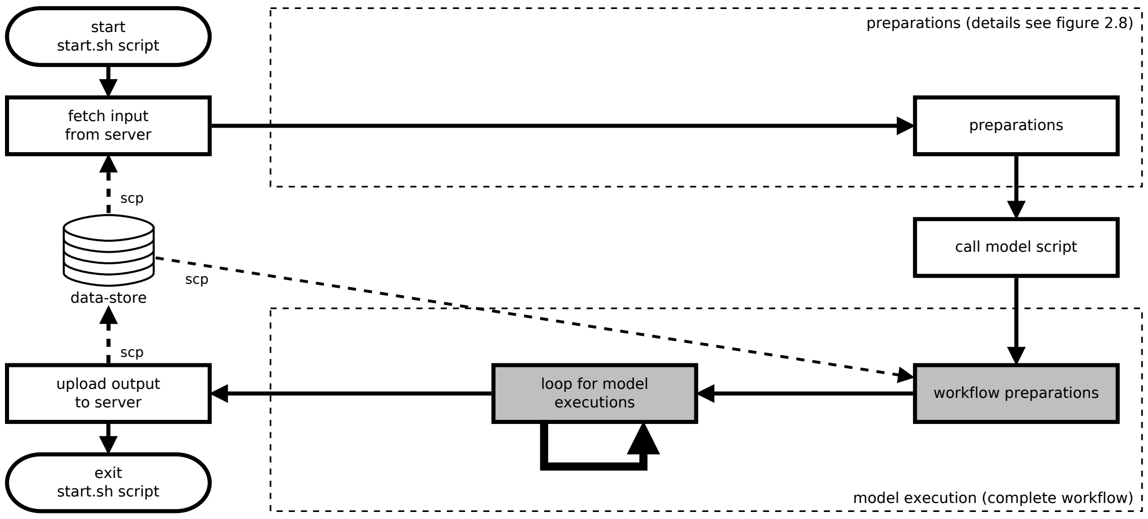


Figure 3.2: Chart of the modularized demoChain, based on figure 2.7

the same directory as the start.sh script. This file contains one model name per line and nothing else.

After the model script parses the specified models from the models.do file into an array the models are categorized in the three model layers like shown before in figure 2.2 on page 7. From each layer only one model can be executed in every workflow. But not all layer must be set and layer 2 can also be skipped or a single model can be specified. To return the sorted order a temporal array with three items is defined with a specified starting value.

For each given model the affiliation to the layers is checked. If a model fits into a layer and the corresponding item in the temporal array (e.g. array[0] for layer 1) has still its origin starting value, the tested model is set on this position. Otherwise the check exits with an error. It also exits if no layer the model belongs into could be found, which implies that the model does not exist or is spelled wrong. This is repeated for each given model in the models.do file.

After all models are checked the temporal array includes up to three ordered models and is returned to the demoChain model script again where it is written back into the initial array. But before the models can be executed an input must be fetched for the first model. For every model a pre-composed input file is available on the data-store. Because the file names have a clear structured syntax no other information for the fetch of the input is necessary. For example the input file for wrf-nmm is simply named *demo.input.wrf-nmm.tgz*.

After the input is located in the job directory the model execution can start running a loop over all given models.

3.2.2 Loop for model executions

As mentioned before it was the primary objective of this thesis to avoid all unnecessary operations. Of course, it was reasonable to develop the model execution in a loop which can exclude as much preparation work as possible. The section before described the required steps to prepare the execution environment. At this point the entire environment is prepared in a way that all models can repeatedly use this setting in the same configuration. Thereby

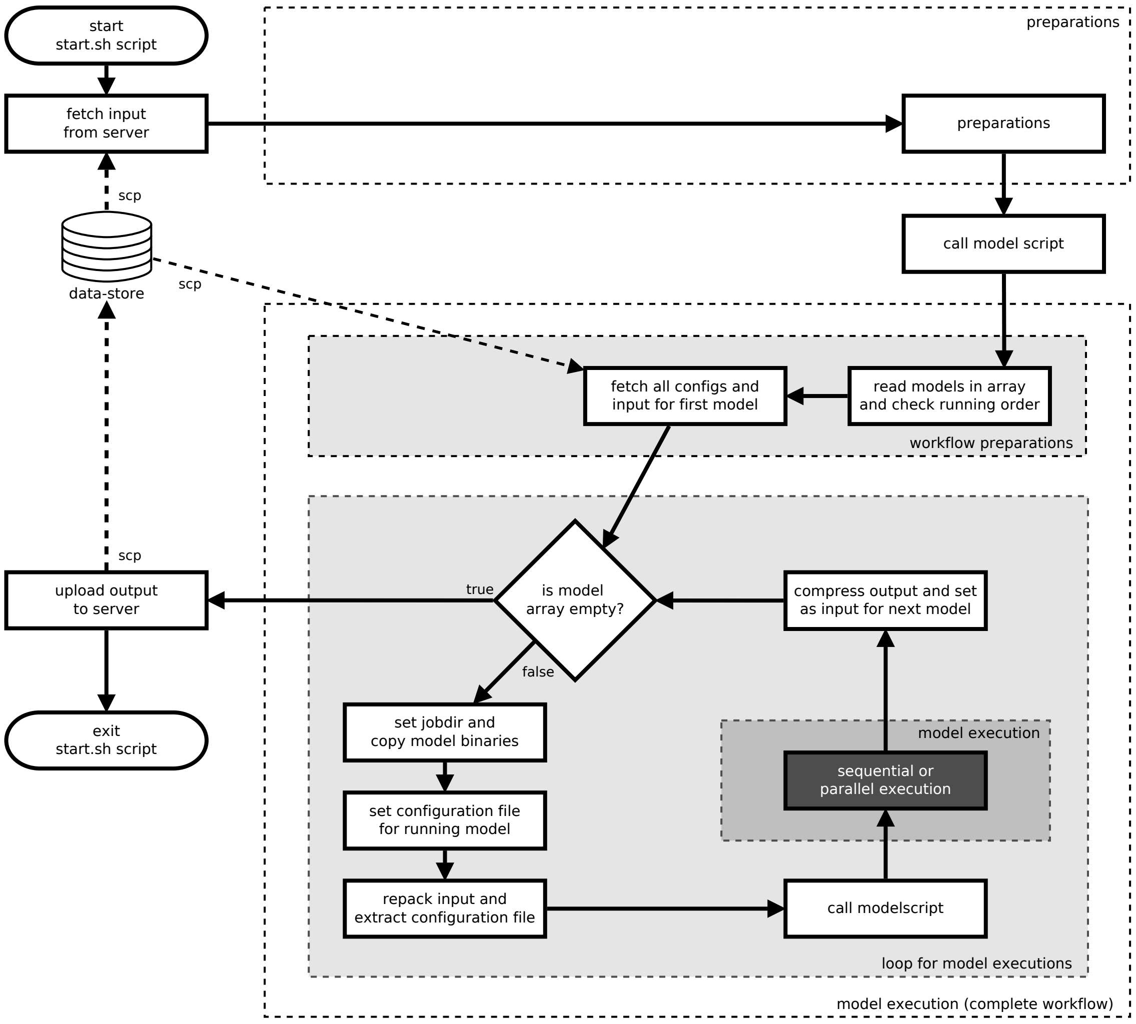


Figure 3.3: Chart of the modularized demoChain, based on figure 3.2

any possible workflow can compute to the end without any changes on the local environment.

To ensure modularization the execution of any model within a workflow has to be a periodic process. In figure 3.3 the complete loop is shown in detail with the white boxes in the gray area named *loop for model executions*. As long as the created array with the ordered models is not empty the loop is entered and is driven by one model. The name of the model is saved to a variable and due to a very good naming scheme in the repository another function can create a temporal working directory named like the model and copy the model binaries there. This is very important like in any other single model runs (cf. section 2.2.2) because all executions of models within a workflow can guarantee that no influences on the repository is made and the correctness of the whole DRIHM computation is not touched. The copied binaries, scripts and other needed files can now be used by the called model script.

Mapping the input All DRIHM models produce an output file which is always a gzipped tar archive. But there is no clear structure regarding the naming of the files and the directory tree creation within DRIHM. The output files from wrf-nmm and wrf-parallel, which in fact contain nfc files of the same data-structure, are named and placed within this archive completely different. Both outputs, from wrf-nmm and wrf-parallel, can be used as an input file for all following models. To ensure that an output from such a model can be used as an input for another a mapping of the archive's content is needed.

The start.sh script has a function implemented to perform this. The needed information is written in a mapping.txt file which contains only one line with the old and the new name and location. This file is gzipped together with the model's configuration files, which include all needed configuration and rainfall or terrain parameters for the model calculation. This differs within the models from one text file to a complete data set of the basin in the region of Genoa.

To ensure that all models can be combined in workflows every possible combination needed to be pre-implemented. All these configuration files of all model combinations are fetched from the data-store (cf. section 3.2.1) as one archive and extracted to the temporal working directory of the demoChain as mentioned before. Because the model names can be addressed in the demoChain script an efficient syntax was implemented, which includes in all configuration file names a combination of the current and, if a prepended model exist, the name of this model. From the demoChain temporal directory the proper configuration file can be easily copied to the model's temporal directory at this point of the loop. For example, if the model drift is called after a run of the model rfarm, the corresponding configurations file is named *configuration.ribs.rfarm.tgz* and can thus be fetched and processed very simply.

Afterwards, the needed configuration file is extracted and the input file is repacked as defined in the mapping file. The mapping function again packs an archive of the input files because this input archive is not unpacked before the model scripts do this. Because this is a specification of the start.sh script it is also applied to the demoChains. All needed files for the model computation are at this point of the loop in the temporal directory of the model and the model can start its actual computation.

Model execution Every model script runs its computation like in every single run. The execution is similar to the grayed boxes in figure 2.8 on page 13. But the input file differs in each run of the loop. Fortunately, the start.sh script innately passes the location of the input file to the model script in a variable. The loop renews the path to the before mapped input file for each model. Here again, no adjustments on the model script were necessary.

The differentiation whether a model is parallel and batched to a scheduler or runs sequentially is also accomplished by the model script itself. At this state of the DRIHM project the only models which compute in parallel are the models from layer 1: wrf-parallel, wrf-nmm and meso-nh. Only they can be queued into a scheduler like shown in figure 3.1. The others run their computation directly on the resource server. When finished, the fetching of the output files and their compression back to an archive is controlled and finished by the model script.

To make an output accessible by the start.sh script, its location is again written to a variable. To continue in the loop the same path is set to the variable of the input file. Afterwards, the loop can start again with a new model or finish.

Exiting the loop When the models' array is empty the workflow is finished. Then the gzipped output of all model computations is collected and gzipped again to a new archive containing all single outputs and the complete workflow output. This is the last action from the demoChain script after which the start.sh script takes the control again on all remaining commands. One of them is that the output archive is uploaded to the data-store. From there it can be accessed and used to debug the models or the complete workflow. Start.sh finishes afterwards with a deletion of all temporal files as explained in section 2.2.2 before and completes the whole run.

The loop includes all necessary steps, which are sufficient for a correct run of every model. All model scripts developed for and included in DRIHM have as a major requirement that no changes to the system environment are permitted. Thus it is not required to reset the environment after a model run. Would a model script change any local variables the finishing start.sh script could also not work correctly in all single runs. It is assumed that this is always controlled by the developers on the one side and the technical authority at LMU on the other. Therefore, this implementation is not responsive to such changes and assumes that the correctness of the local environment stays valid.

3.2.3 The modularized demoChain

demoChain01: all possible workflows With the guarantee of a correct and stably implemented new model the demoChain was added to the DRIHM framework with the model name *demoChain01* and can be run from the same gateways like any other model. At the current state of DRIHM it was not possible to chain any combination with the model *meso-nh* because no configuration and no input files could be prepared. Furthermore, the workflow *wrf-parallel - ribs* leads to an error precipitated by an incompatibility of the two models in the DRIHM framework. This problem should be solved by the developers within the scope of the project.

3.3 Pre-defined model chains in DRIHM

This thesis has the objective to implement two pre-defined model chains for DRIHM. This task was extended by the concept of the modularized chain presented in the last sections instead of hard coded script files. Due to the introduction of a simple file to compose the models the pre-defined chains arise from only few lines of code. With the specification of the models to be calculated in the *models.do* file by writing them there one model per line, two chains were added. They cover the most important models within DRIHM. All implemented demoChains are listed in figure 3.4 including the modularized demoChain01 for all workflows.

Data set for the demoChains All calculations in the demoChains base on the 2011 flood event in Genoa. Within DRIHM the basin around Genoa and the recorded rainfall values are available. They have been integrated in the front end and can be configured by the user. For this thesis all models were configured manually in the front end. The resulting configuration files were transferred to the back end and repacked to be used in all possible workflows as shown.

To avoid too long computations in one model the configurations were set to minimize the running time. The main time period for all models is set to a six hour period, from midnight

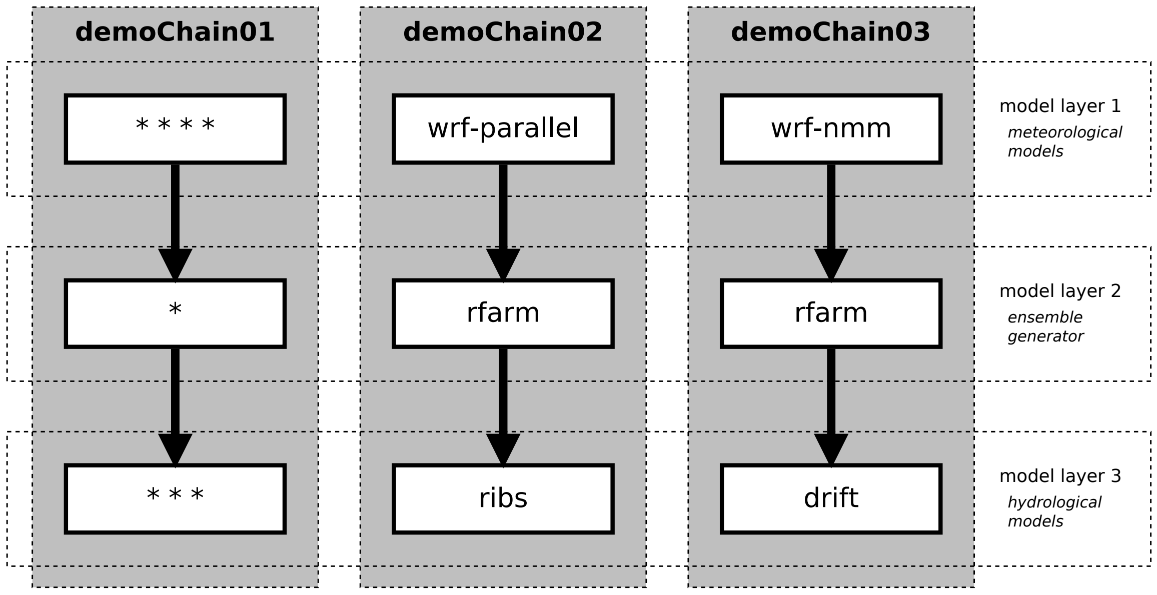


Figure 3.4: Chart of the modularized and the two pre-defined model chains

to six o'clock on November 4th, 2011. In conjunction with this, the result can certainly not produce any authoritative results for real flood predictions, but is sufficient to show any user the functionality of the DRIHM models and that the result is correct. Also the physical configurations were limited and set to a minimum. The period of 6 hours could not be chosen shorter because some models require this as a minimum input time to compute without an error and to get reasonable results. Within these limitations and prospects the existing configuration and input files represent all important components of DRIHM in its current state.

demoChain02: wrf-parallel - rfarm - ribs This demoChain consists of three models, one from each layer. The workflow bases on a pre-calculated input file, including the mentioned data sets of the flood in Genoa for the *wrf-parallel* calculation. To the model *rfarm* some required files need to be added, e.g. the observed rain values in the correspondent time, which are prepared in the respective configuration file.

The last model *ribs* finishes this workflow by issuing river discharge levels at six locations in the suburban area of Genoa. Due to a bug in one of the contributing models, the result of the workflow is computed without any error, but only with discharge levels of zero cubic meters per time. This was reported to the developer but has not been solved at the moment of writing.

The complete code of the demoChain02 script, which is called by start.sh, is very short and simple, because it is based on the modularized demoChain01, and is shown in listing 3.1. The demoChain02 script executes demoChain01 after creating the models.do file.

This file is created in the *mydir* directory, the root directory of any DRIHM computation on any resource where also start.sh is located. After the three models, which are to be executed, are written into the models.do file, the demoChain01 is called within the same bash (due to the dot before the script name) without any parameter. The called script expects the models.do file and reads the specified models from there. It is the same call

Listing 3.1: demoChain02 code

```
MODELSDO="$MYDIR/models.do"

echo "wrf-parallel"      >> $MODELSDO
echo "rfarm"             >> $MODELSDO
echo "ribs"              >> $MODELSDO

. $MODELDIR/demoChain01.sh
```

start.sh would run the demoChain01 if selected. Afterwards, the so composed workflow will be executed as detailed in section 3.2.

The code of demoChain03 is similar to the shown code of demoChain02, only with changed model names in the models.do file, and can be seen in the appendix 3.

demoChain03: wrf-nmm - rfarm - drift The last demoChain is also a workflow containing three models, again one from all three layers. Like the demoChain02, its input covers the same time period, but is calculated especially for *wrf-nmm*. After the ensemble generator *rfarm*, the last model is *drift*. Its output includes several files, but the most interesting is the development of the river discharge level at the footbridge Firpo in Bisagno, a quarter of Genoa. The progress of this destructive flood predicted by this demoChain workflow can be seen in figure 3.5 in the first graph *a*. This visualization is created by a tool which is integrated into the DRIHM portal and can be used to observe any other result.

Also ribs produces output files which can be visualized. The growth of the discharge level at two different locations can also be seen in figure 3.5, in the graphs *b* and *c*. The time period of these graphs is shorter, because no data from the model rfarm was added.

As shown in the code of demoChain02 before, it is assured that a composition of any other not pre-defined chain is very simple and can be done by every authorized DRIHM user in the back end. It is only necessary to create a models.do file containing one model per line in the same directory as the start.sh is placed.

3.4 Results summary

This chapter introduced a new model, which was added to the DRIHM framework, to run complete workflows with up to three models in one call. For every run no additional information is needed, in particular no configuration or input files. They are all pre-calculated and available for any model combination.

The complete implementation code can be seen in the appendix 1 of this thesis. In the sections before it was shown which measures needed to be realized to enable such an economic and easy to understand solution of the given problem, which were in detail unnecessary data transfers, avoidable scheduler waiting times and a complete non-realization of workflows in the back end. This also ergonomic implementation adds a new model script, which only needs the names of the models within a text file, to run a correct and exemplary workflow.

The modularized demonstration chain model demoChain01 and the pre-defined ones demoChain02 and demoChain03 are already part of the DRIHM framework. They can be

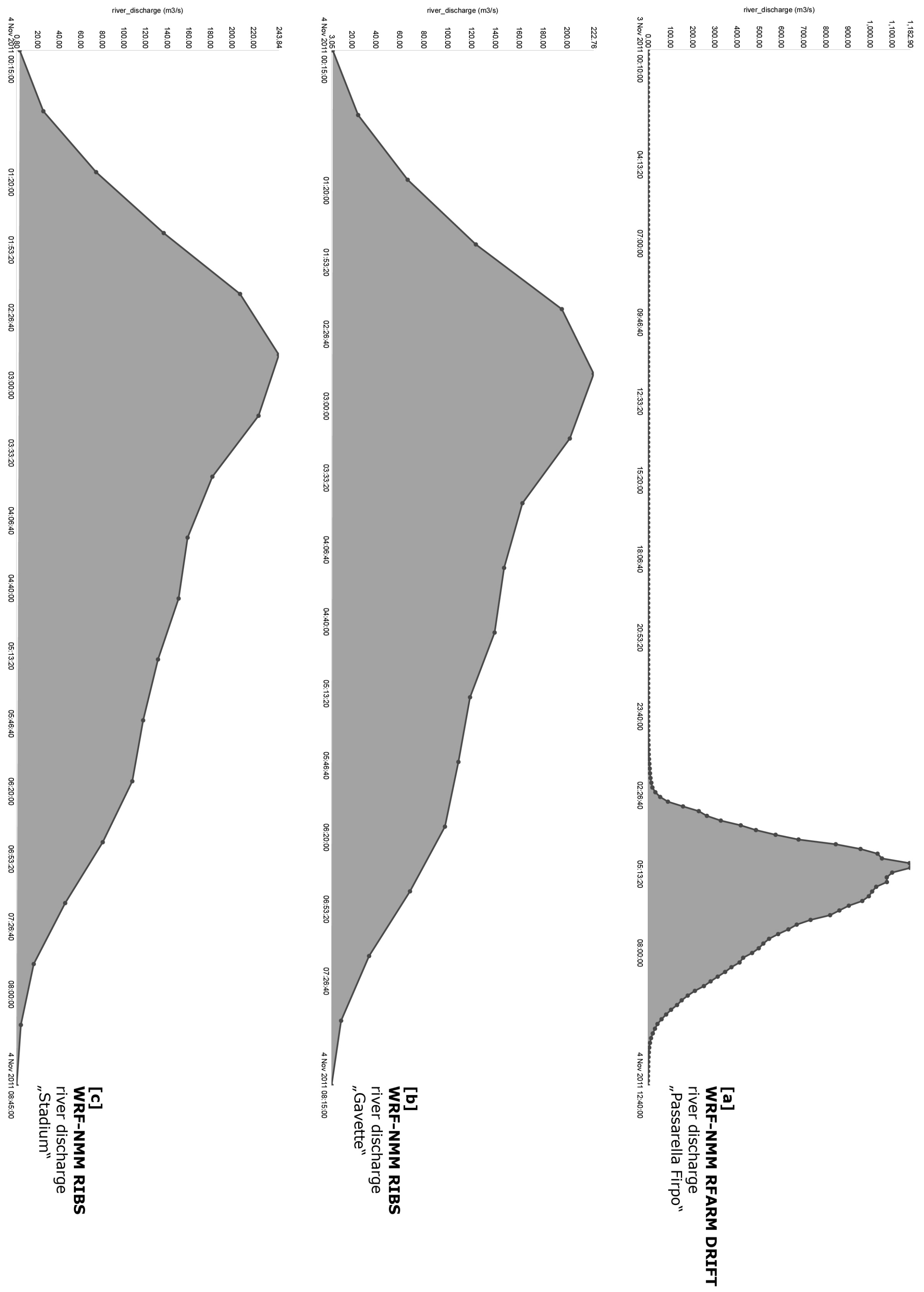


Figure 3.5: Output-visualization from the demoChain workflow *wrf-nmm rfarm drift* and *wrf-nmm ribs*, designed in the DRIHM front end [DRI15c]

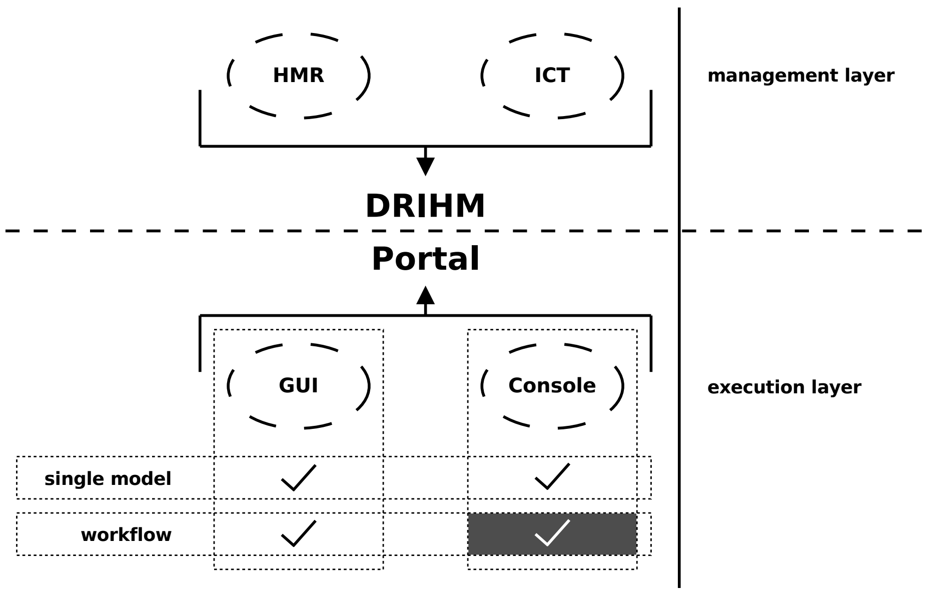


Figure 3.6: The solved problem statement of this thesis based on figure 1.1

started from any back end server to illustrate the possibilities of the DRIHM project by composing different models to a workflow or run a single model to get a specified insight.

As told in the sections before, all valid combinations are covered by this solution (except the mentioned models, which cannot be combined in its current state in section 3.2.3). For all these workflows and single runs configuration and input files were added to the data-store and can be accessed from any execution of the start.sh script on any resource, also worldwide.

The next chapter will discuss and evaluate the concept as well as its implementations and will also respond to possible challenges which come with this development.

4 Summary

Chapter 3 introduced a new model into the DRIHM framework, which renewed the current state of composing models to a workflow with a concept that allows all models to be executed within one call and within one transmission to one resource. The achieved speed up of current workflows was done within the requirements of the start.sh script which forced to use the given environment and did not allow any serious changes to start.sh and to the existing model scripts.

This chapter will give a summary and evaluate the concept given in section 3.1 regarding the main facts in section 4.1. Afterwards, in section 4.2 the findings of this thesis are discussed, focused on the compatibility with the whole DRIHM framework now and in future.

As pointed out, every workflow composed in the DRIHM portal was separated in single model runs (cf. figure 2.9 on page 15). This thesis had the goal to form a possibility to run a complete workflow within one submission, what succeeded with the given concept. This is not only possible in the back end of DRIHM, it could also be integrated into the front end as a fast running, pre-defined demonstration case in the DRIHM portal. Therefore, this thesis offers a complete framework with configuration files for each single and composed run, as well as the required input files. If integrated into the portal, such a pre-defined workflow could give inexperienced users an understanding of the possibilities of chaining models within the DRIHM project.

4.1 Evaluation

The presented concept is one possible solution to approach the given problem. Of course, this needs to be considered in several views. In the last chapters different perceptions were mentioned in which performance improvements could be achieved. They are summarized in this section.

Data set At first sight, it is very comfortable to have pre-defined configuration and input files. In any execution of the demoChain no configuration files or input adaptions must be considered. They are all available and processed automatically. But due to the fact that they are pre-defined it is not possible to change anything in one configuration file without verifying that all others are still correct. Especially the time period is hard coded for all model computations within the demoChains and needs to be changed very carefully. Also the basin is determined and set to Bisagno in Genoa. An adaption to any other region would change all setting files. This is certainly possible but associated with a lot of configuration work.

Maintenance One objective of the concept is to provide a simple maintenance feasibility, to test all models on impacts regarding their operativeness when changing the environment or other models. This is very important and the demoChain is a real improvement. But to

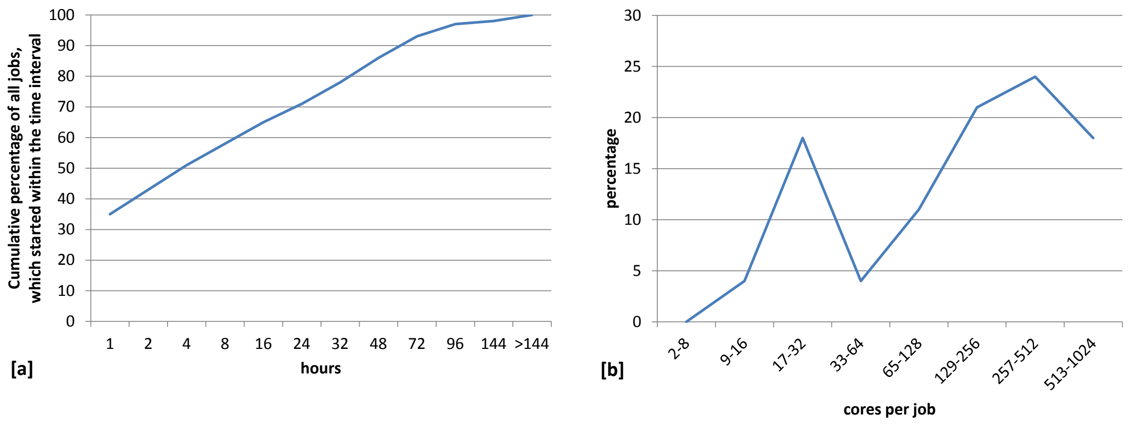


Figure 4.1: Average queueing time of submitted jobs at the LRZ MPP-cluster[1] in 2014, provided by LRZ

guarantee the correctness of this test, the enclosed configuration and input files need to be updated if the appropriate model is affected by an update. This must always be taken into account. The relevant files are saved in the data-store, but any modification should always be done by the developers of the depending model directly, because they know their models in details and need the same files for their test stage in any case.

Queueing time Another improvement is the saving of time if not entering the grid resource scheduler several times. Reducing this from three times to one is obviously an enhancement, including the connected authorization problems based on the certificates' life-times.

The queueing time improvement would be maximized if also the scheduler times of the DRIHM model executions could be modified. But this was not considered within this thesis. Although, all parallel models would enter a separate queue on the corresponding cluster, it is not an enhancement in the current state of DRIHM to combine all models on one resource, because only the models of layer 1 are parallelized and only they need more than one node allocation on the clusters.

A modification of all workflows is not reasonable before more models run in parallel. Then, based on an e-infrastructure that includes shared resources, which are really used from different scientists and projects and thereby having a relevant queueing time, it would be of course an improvement to allocate computing nodes only once. In figure 4.1 *a* the average queueing time of submitted jobs at the LRZ MPP-cluster[1] in 2014 is summed up in a diagram. Including the right part *b* of the figure, it becomes apparent, if more nodes are required, the waiting time increases up to several days, which of course will also apply to the DRIHM models in future. In this case an improvement of the current situation is unavoidable.

But until then, the current infrastructure prefers the current realization for the huge computation workflows. Because the models differ completely in their performance environment, the DRIHM e-infrastructure includes dedicated servers for certain models. The

[1]Die entsprechende Infrastruktur (das MPP-Cluster) wurde 2015 im Rahmen eines DFG-Großgeräteantrags ersetzt, um die langen Wartezeiten zu minimieren.

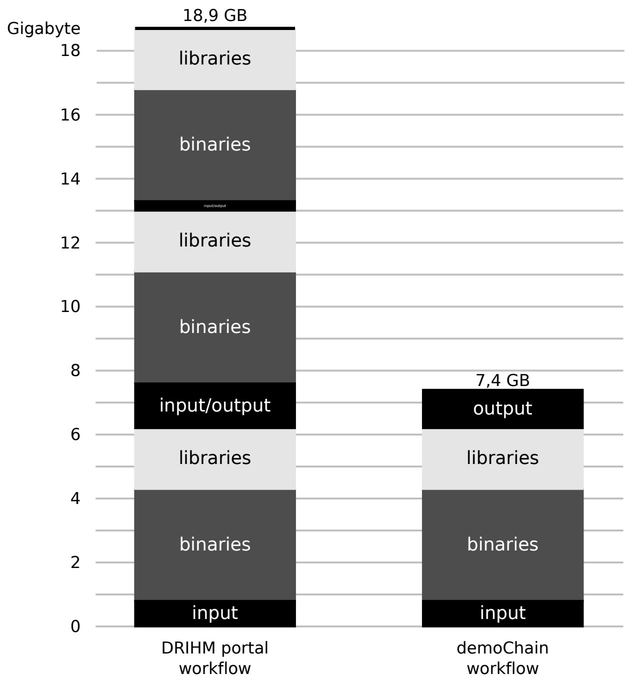


Figure 4.2: Transferred data in a DRIHM portal workflow in comparison with the same workflow as a demoChain

non-parallelized models are not submitted to the powerful clusters in the grid, which are not built for sequential computations. The models calculate on smaller solutions.

So it is a substantial consideration how and especially where the models are submitted. The current realization in DRIHM is an agreement over all participating institutions. The developed concept in this thesis aims to be a contribution to further enhancements and can improve the whole DRIHM performance if the development pursues such a direction.

Traffic reduction Although it might not be possible to transfer this concept to the whole DRIHM structure at the moment, it should be taken note of the opportunities to reduce the real huge amount of traffic. This needs to be taken into account more and more if other resources are added to the infrastructure and the synchronization is no synchronization any more, but a real transfer with every submission of a job. This is not unlikely because it cannot be assumed that all clusters can provide an static local storage and thus, depending on the connectivity of the cluster, the transfer time can be huge compared to the running time of the model. This would obviously make any computation slow and expensive.

In figure 4.2 the transferred data of a three model workflow from the DRIHM front end is compared with the same workflow composed as a demoChain. As shown, the portal workflow transfers three times the data which needs to be transferred in the optimized demoChain. In detail, these are three times 3.5 gigabyte for the binaries and 1.9 gigabyte for the libraries, so totally 16.2 gigabyte only for the DRIHM files.

Within the demoChains, the file sizes of the input files are chosen very small to be economical, but in extensive cases the input and the output files increase up to several gigabyte, which are computation volumes these models are designed for. In the example in figure 4.2

the transferred input and output files in the DRIHM portal workflow had a size of 2.6 gigabyte and in the to this input adapted demoChain, with the same first input file, only 2.0 gigabyte. This economy depends on the fact, that all intermediate results are only transferred to the data-store server at the end of any demoChain and not fetched again for every following model, but stored locally during the complete workflow.

The saved amount of transferred data only for this workflow is about 11.5 gigabyte, what is equivalent to a reduction of 62 percent. Also the number of file transfers is reduced from six to two, and from three synchronization calls to one.

Portability The implemented demoChain could be successfully tested in a new, not in the DRIHM e-infrastructure integrated production system, located at the University of Virginia, in the context of DRIHM2US. Another EU-funded project, which fortifies the cooperation between Europe and the USA for the development of a HMR e-infrastructure and thus ensures the permanent and effective availability of data and models across scientific disciplines and particularly across the Atlantic [DRI15a]. Instead of the via EGI or PRACE used clusters in Europe, this cluster was accessed with the grid middleware Genesis II, on *Extreme Science and Engineering Discovery Environment* (XSEDE) developed by the Virginia Center for Grid Research [XSE15], which is not integrated in the DRIHM framework yet and so all jobs had to be submitted manually to test the demoChain there.

A workflow of *rfarm - drift* could be computed without any changes to the demoChain or start.sh code. This is of course a success of the design of the start.sh script and the complete DRIHM project. But it became apparent that DRIHM computations, which need no additional files and parameters with the execution call like the demoChain, are very simple to realize in a not directly accessible environment like this US cluster. So instead of transferring configuration files for any test in a complicate way through the used middleware, the demoChain fetches the needed files simply itself from the data-store and enable so quick executions on foreign clusters.

4.2 Summary and outlook

In summary, this writing shows another way to submit HMR models, joined as a DRIHM workflow to one grid resource in one transmission and added therefore three models to the DRIHM framework.

The roundup of the evaluation points out two facts: the concept of this thesis offers on the one side a new kind of workflow submission, but on the other site it cannot be integrated into the whole the DRIHM project in its current state. Although the EU project is well-thought-out, its infrastructure can still be improved. This might happen, when the complete project is reviewed and new recommendations influence the progress of the project. Then an adaption of the scheduler process to execute the models might be also reviewed and parts of the developed concept might be integrated.

Changes on this concept of the demoChain and its implementation are also possible and possibly helpful. The requirements and specifications of start.sh, on which all model scripts are based, will always be a limitation. A complete restructuring of the DRIHM project is of course not reasonable.

But a simple improvement could be implemented very fast if the models would be grouped corresponding to their execution type regarding a sequential or parallel execution and depending on how many nodes the execution needs. Afterwards one node allocation and one job submission per group would improve the named facts in section 4.1, especially if more models would run in parallel. This adjustment would be very easy to realize through all models and could improve the performance of all workflow computations noticeably.

The implemented demoChain would be ready to be integrated as a real demonstration chain into the DRIHM portal. With the pre-defined workflows the demoChain is very suitable for any kind of manual or simple demonstration on how models can be composed without worrying about any configuration details.

Going one step further, some configuration parameters of the demoChain concerning model details could be extended from the pre-defined packages and included into the DRIHM portal. This would be an enhancement for any user, who could then make changes on the results of the demoChain workflows, but still without configuring a complete model chain from scratch.

However, with the further development of DRIHM and especially the integration of the not yet included models, this demoChain might not be able to cover still all models. Because the concept is based only on models, which can be executed on one resource with one execution environment, no model which needs other specifications could be added to the demoChain framework. But due to an already integrated function, which checks the given demoChain models, this would not have any influence on the correctness of the demoChain.

List of Figures

Bibliography

[BPQ+12] BEDRINA, T., A. PARODI, A. QUARATI, A. CLEMATIS et al.: *ICT approaches to integrating institutional and non-institutional data services for better understanding of hydro-meteorological phenomena.* Natural Hazards and Earth System Science, 12:1961–1968, 2012.

[DCG+14] DAGOSTINO, D., A. CLEMATIS, A. GALIZIA, A. QUARATI et al.: *The DRIHM Project: a Flexible Approach to Integrate HPC, Grid and Cloud Resources for Hydro-Meteorological Research.* SC14: International Conference for High Performance Computing, Networking, Storage and Analysis, pages 536–546, 2014.

[DRI15a] DRIHM2US.EU: *Distributed Research Infrastructure for HydroMeteorology to United States of America (DRIHM2US)*, June 2015. `http://www.drihm2us.eu/images/documents/RI-313122-DRIHM2US_FACT-SHEET.pdf`.

[DRI15b] DRIHM.EU: *D5.2: Report on the Inventory of Deployed Services*, May 2015. `http://www.drihm.eu/images/Deliverable/drihm-dwp5.2-20120228-v2-lmu-report_on_the_inventory_of_deployed_services.pdf`.

[DRI15c] DRIHM.EU: *DRIHM front end web portal*, June 2015. `http://portal.drihm.eu/`.

[DRI15d] DRIHM.EU: *Objectives of the DRIHM e-Science environment*, April 2015. `http://www.drihm.eu/index.php/project/objectives/`.

[DRI15e] DRIHM.EU: *Schematic diagram of the forecast chain for floods*, April 2015. `http://www.drihm.eu/images/chain.bmp`.

[EGI15] EGI.EU: *EGI-DRIHM:Collaboration*, June 2015. `https://wiki.egi.eu/wiki/EGI-DRIHM:Collaboration/`.

[HCG+15] HALLY, A., O. CAUMONT, L. GARROTE, E. RICHARD et al.: *Hydrometeorological multi-model ensemble simulations of the 4 November 2011 flash flood event in Genoa, Italy, in the framework of the DRIHM project.* Natural Hazards and Earth System Sciences, 15:537–555, 2015.

[LRZ15] LRZ.DE: *Leibniz Supercomputing Centre: Overview of the Cluster Configuration*, April 2015. `http://www.lrz.de/services/compute/linux-cluster/overview/`.

[ND15] NGI-DE.EU: *German National Grid Initiative*, April 2015. `http://www.ngi-de.eu/`.

[Sch15] SchedMD.com: *Slurm Workload Manager*, April 2015. `http://slurm.schedmd.com/`.

[VA12] Vachhani, Milan K. and Kishor H. Atkotiya: *Globus Toolkit 5 (GT5): Introduction of a tool to develop Grid Application and Middleware.* International Journal of Emerging Technology and Advanced Engineering, 2:174–178, 2012.

[XSE15] XSEDE.org: *Extreme Science and Engineering Discovery Environment (XSEDE) - Genesis II*, June 2015. `https://portal.xsede.org/knowledge-base/-/kb/document/bbfb/`.

demoChain source code

1 demoChain01

Listing 1: demoChain01 model script

```
#################################
#                               #
# modularized demoChain script #
#                               #
#################################

###
# functions runModel, checkChain
###

#
## runModel creates the jobdir, copies the binaries, sets the
## configuration file, repacks the input and executes the model
#
function runModel {
  #Setting JOBDIR for $1-execution!
  if [[ $1 == "wrf-parallel" ]] ; then
    JOBDIR=$ORIGJOBDIR/wrf
  else
    JOBDIR=$ORIGJOBDIR/$1
  fi

  # copy binaries to JOBDIR
  if [[ $DEBUG -ge 1 ]] ; then echo "Preparing Enviroment for $1
     ... "; fi
  if [[ $1 == "wrf-parallel" || $1 == "wrf-nmm" ]] ; then
     ORIGMYDIR=$MYDIR
     MYDIR=$ORIGJOBDIR
     prepare-environment-cp-bin $1
     MYDIR=$ORIGMYDIR
  else
     prepare-environment-cp-bin $1
  fi
  if [[ $DEBUG -ge 1 ]] ; then echo "done."; fi

  # copy the correct configuration file in MYDIR, where it is
     expected from repackInput
  if [[ $DEBUG -ge 1 ]] ; then echo "Preparing configuration file
      for $1 (after $2)... "; fi
  if [[ $1 == $2 ]] ; then
    cp $ORIGJOBDIR/configuration.$1.tgz $MYDIR/configuration.tgz
    if [[ $DEBUG -ge 1 ]] ; then echo "$ORIGJOBDIR/configuration.$1.tgz $MYDIR/configuration.tgz"; fi
  else
    cp $ORIGJOBDIR/configuration.$1.$2.tgz $MYDIR/configuration.tgz
    if [[ $DEBUG -ge 1 ]] ; then echo "$ORIGJOBDIR/configuration.$1.$2.tgz $MYDIR/configuration.tgz"; fi
  fi
  # extract configuration file and repack input if necessary
  repackInput
  if [[ $DEBUG -ge 1 ]] ; then echo "done."; fi

  # start execution of model: call model script
  if [[ $DEBUG -ge 1 ]] ; then echo "Starting execution of $1... "; fi
  . $MODELDIR/$1.sh
  if [[ $DEBUG -ge 1 ]] ; then echo "done."; fi

  if [[ $DEBUG -ge 1 ]] ; then echo "Collecting $1-output... "; fi
  # save output as original file and for the next model as INPUTFILE
  cp $OUTPUTFILE $INPUTFILE
  cp $OUTPUTFILE $ORIGJOBDIR/output.$1.tgz
  if [[ $DEBUG -ge 1 ]] ; then echo "done."; fi

}

#
## checkChain checks given models and returns an ordered array
#
function checkChain {

  if [[ $DEBUG -ge 1 ]] ; then echo "Checking models for correct chaining... "; fi
  LAYER1=(wrf wrf-parallel wrf-nmm meso-nh)
  LAYER2=(rfarm no-ensembler)
  LAYER3=(ribs drift hbv)
  ALLOWED=1

  # define function to print syntax information for models.do file
  function printChaining {
    echo "ERROR: specified models can not be combined!"
    echo " "
    for model in $(seq 0 $((${#MODELS[@]} - 1)))
      do
      echo "  ${MODELS[$model]}"
    done
    echo " "
    echo "Please compose models as shown:"
    echo " "
    echo "  LAYER1 [ wrf | wrf-parallel | wrf-nmm | meso-nh ]"
  echo "  LAYER2   -> [ rfarm ]"
  echo "  LAYER3     -> [ ribs | drift | hbv ]"
  echo " "
  echo "Maximum 1 model from each layer. Please change models in models.do file!"
  exit -1
}

# helping function to search an object in an array
function contains () {
  local seeking=$1; shift
  local in=1
  for element; do
    if [[ $element == $seeking ]]; then
      in=0
      break
    fi
  done
  return $in
}

# define array to store the order of models
# Index 0 is first model ...
TMPMODELS=([0]=none [1]=none [2]=none)

# Test all models from models.do if a model from the same layer is already
# set. If not, set the model in the array, so this position is blocked.
# exit through printChaining if a model is unknown or layer is already set.

for model in $(seq 0 $((${#MODELS[@]} - 1)))
  do
  if [[ $DEBUG -ge 1 ]] ; then printf "Sorting ${MODELS[$model]} in chain"; fi
  if $(contains ${MODELS[$model]} "${LAYER1[@]}" ) ; then
    if [[ "${TMPMODELS[0]}" == "none" ]]; then
      TMPMODELS[0]=${MODELS[$model]}
      printf " - chaining OK\n"
    else
      printf " - ERROR: already chained ${TMPMODELS[0]} in LAYER1\n"
      printChaining
    fi
  elif $(contains ${MODELS[$model]} "${LAYER2[@]}" ) ; then
    if [[ "${TMPMODELS[1]}" == "none" ]]; then
      TMPMODELS[1]=${MODELS[$model]}
      printf " - chaining OK\n"
    else
      printf " - ERROR: already chained ${TMPMODELS[1]} in LAYER2\n"
      printChaining
      fi
    elif $(contains ${MODELS[$model]} "${LAYER3[@]}" ) ; then
      if [[ "${TMPMODELS[2]}" == "none" ]]; then
        TMPMODELS[2]=${MODELS[$model]}
        printf " - chaining OK\n"
      else
        printf " - ERROR: already chained ${TMPMODELS[2]} in LAYER3\n"
        printChaining
      fi
    else
      printf " - ERROR: model does not exist\n"
      printChaining
    fi
  done

  # Set new array with correct order as new MODEL array and remove nones

  if [[ $DEBUG -ge 1 ]] ; then
    printf "Starting Execution of chained models:"
    none=( none )
    MODELS=(${TMPMODELS[*]/$none})
    for model in $(seq 0 $((${#MODELS[@]} - 1)))
      do
      printf "  ${MODELS[$model]}"
    done
    echo " "
  fi
}

###
# prepare models' array, configurations and first input
###


# Fetch configurations files from Server
if [[ $DEBUG -ge 1 ]] ; then echo "Preparing Configurations for $1... "; fi
cd $JOBDIR
myCopy "FETCH" "demodata/demo.configurations.tgz"

# Extract configuration files in $JOBDIR
tar -xzf $INPUTFILE
if [[ $DEBUG -ge 1 ]] ; then echo "done."; fi

# read models.do file in an array, one line, one model
MODELFILE=$MYDIR/models.do
if [[ ! -r $MODELFILE ]] ; then
  echo "ERROR: expected models.do does not exist or is not readable!"
  echo "Please put the >models.do< file in $MYDIR and make it available!"
  exit -1
fi
MODELS=( $(cat "$MODELFILE") )
if [[ ${#MODELS[*]} < 1 ]] ; then
  echo "ERROR: models.do does not include a model; it's empty!"
  echo "Please write one model per line in the >models.do< file!"
  exit -1
fi

# sort models in a correct order and check if all models exist
checkChain

# The first models needs an input, which is fetched here
if [[ $DEBUG -ge 1 ]] ; then echo "Getting input for ${MODELS[0]}..."; fi
myCopy "FETCH" "demodata/demo.input.${MODELS[0]}.tgz"
if [[ $DEBUG -ge 1 ]] ; then echo "done."; fi

###
# execute the model
###

# Save original JOBDIR to write it back
ORIGJOBDIR=$JOBDIR

# Save the last executed model to get later the correct configuration file
PREVMODEL=${MODELS[0]}

# starting the loop, executing model after model
for model in $(seq 0 $((${#MODELS[@]} - 1)))
  do
  if [[ $DEBUG -ge 1 ]] ; then echo "$1: Starting ${MODELS[$model]}"; fi
  runModel ${MODELS[$model]} $PREVMODEL
  PREVMODEL=${MODELS[$model]}
  if [[ $DEBUG -ge 1 ]] ; then echo "${MODELS[$model]} finished."; fi
done

#
# generate output-file
#

# Set original JOBDIR and collect output archives
if [[ $DEBUG -ge 1 ]] ; then echo "Collecting Output from $1... "; fi
JOBDIR=$ORIGJOBDIR
cd $JOBDIR
tar -czf $OUTPUTFILE output.*
if [[ $DEBUG -ge 1 ]] ; then echo "done."; fi

# Output is uploaded in start.sh to server
```

2 demoChain02

Listing 2: demoChain02 model script

```
################################
#                              #
# demoChain02 script           #
#                              #
################################

#
## Writing pre-defined models in models.do file
#

if [[ $DEBUG -ge 1 ]] ; then echo "Setting up models.do file..."; fi

TMPMODELS="$MYDIR/models.do"

echo "wrf-parallel"     >> $TMPMODELS
echo "rfarm"            >> $TMPMODELS
echo "ribs"             >> $TMPMODELS

if [[ $DEBUG -ge 1 ]] ; then echo "done."; fi
if [[ $DEBUG -ge 1 ]] ; then cat $TMPMODELS; fi

#
## run demoChain02 via demoChain01
#

. $MODELDIR/demoChain01.sh
```

3 demoChain03

Listing 3: demoChain03 model script

```
################################
#                              #
# demoChain03 script           #
#                              #
################################

#
## Writing pre-defined models in models.do file
#

if [[ $DEBUG -ge 1 ]] ; then echo "Setting up models.do file..."; fi

TMPMODELS="$MYDIR/models.do"

echo "wrf-nmm"           >> $TMPMODELS
echo "rfarm"             >> $TMPMODELS
echo "drift"             >> $TMPMODELS

if [[ $DEBUG -ge 1 ]] ; then echo "done."; fi
if [[ $DEBUG -ge 1 ]] ; then cat $TMPMODELS; fi

#
## run demoChain03 via demoChain01
#

. $MODELDIR/demoChain01.sh
```