Transparent Model-Driven Provisioning of Computing Resources for Numerically Intensive Simulations

Fabian Korte[1], Alexander Bufe[2], Christian Köhler[3], Gunther Brenner[2], Jens Grabowski[1], and Philipp Wieder[3]

[1] University of Goettingen, Institute of Computer Science
37077 Goettingen, Germany
[2] Clausthal University of Technology, Institute of Applied Mechanics
38678 Clausthal-Zellerfeld, Germany
[3] Gesellschaft fuer wissenschaftliche Datenverarbeitung mbH Goettingen (GWDG)
37077 Goettingen, Germany

Abstract. Many simulations require large amounts of computing power to be executed. Traditionally, the computing power is provided by large high performance computing clusters that are solely built for this purpose. However, modern data centers do not only provide access to these high performance computing systems, but also offer other types of computing resources e.g., cloud systems, grid systems, or access to specialized computing resources, such as clusters equipped with accelerator hardware. Hence, the researcher is confronted with the choice of picking a suitable computing resource type for his simulation and acquiring the knowledge on how to access and manage his simulation on the resource type of choice. This is a time consuming and cumbersome process and could greatly benefit from supportive tooling. In this paper, we introduce a framework that allows to describe the simulation application in a resource-independent manner. It furthermore helps to select a suitable resource type according to the requirements of the simulation application and to automatically provision the required computing resources. We demonstrate the feasibility of the approach by providing a case study from the area of fluid mechanics.

1 Introduction

Scientific simulations are often computation intensive and time consuming and can highly profit from choosing a suitable computing resource type and scale. However, choosing the right computing resource and an appropriate scale is not a trivial task, especially in modern computing centers that offer access to a heterogeneous infrastructure including cloud services, high performance computing clusters and clusters with specialized accelerators (e.g., GPU cards). All of these different resource types have their own technical peculiarities and require the user of the simulation application (the *simulation scientist*), to invest time to

learn when and how to use them. It might even be necessary to switch the resource type and scale during the lifetime of a simulation, e.g., when transitioning from testing simulation code to running parameter studies, or when the problem size increases and suddenly requires more computing resources. To overcome this burden, we develop a transparent integration mechanism for heterogeneous computing resources. The goal is to semi-automatically deploy and execute simulation applications on the most suitable resource type. To achieve this goal, we model the simulation application structure and behaviour in a resource-agnostic way and provide model transformators that automatically transform the abstract simulation application model into resource-specific models. In this paper, we introduce a conceptual framework that implements the concept of model-driven provisioning of computing resources for simulation application and provide a case study on the application of the framework by modeling and deploying a simulation from fluid mechanics.
The specific problems addressed in this work are:

P1 Developing and/or using a simulation application and making that application run on various resource types are in principle orthogonal tasks. However both of these tasks are handled by the simulation scientist in practice.
P2 The technical details which need to be learned in order to deploy the same application on heterogeneous infrastructures burden the simulation scientist with a significant investment of time that would be better spent on the application or its use case itself.
P3 Even once several resource types have been shown to be compatible with a certain application, choosing the optimal compute resource for a given job in terms of hardware equipment and available software packages remains highly non-straightforward.

The remainder of this paper is structured as follows: In Section 2, we introduce the simulation application that serves as a use case in scope of this work. After that, we discuss the conceptual framework, we propose to homogenize the utilization of the heterogenous infrastructure in Section 3, followed by a short introduction to the modeling language *Topology and Orchestration Specification for Cloud Applications* (TOSCA), which we use as a basis for building the models in scope of the framework in Section 4. In Section 5, we discuss the modeling of the use case application with help of the framework, followed by a discussion of our findings and the limitations of the approach in Section 6. Finally, we provide an overview of related work in Section 7 and draw our conclusions and give an outlook on future work in Section 8.

2 Use Case

As an exemplary use case, the simulation of the flow in porous media with the *Lattice-Boltzmann method* (LBM) was chosen. LBM originates from Boltzmann's kinetic molecular dynamics and may be understood as a discretization in space and time of the velocity-discrete Boltzmann equation. Its main advantages are

the inherent parallelism which leads to great performance on many architectures and easy handling of complex geometries.
Particle filled beds are of great technical importance e.g., in catalysator packings. Thus, the knowledge of the pressure drops in such packings is crucial. Confining walls can have a significant influence on the pressure drop and therefore the pressure drop as a function of the sphere diameter to wall distance ratio is systematically studied. A high number of packings is created using an algorithm and a scale-resolving simulation of the flow in the packing with LBM is performed (Figure 1). In previous work, measured and simulated pressure drops in slit-type milli-channels with different packings were compared. Very good agreement between measured and simulated pressure drops was achieved [8].
This use case can be seen as a typical parameter study which are common in engineering. The same program is plurally started with different parameters (in this case sphere diameter to wall distance ratio). The computational cost of a single program run is relatively low and the high effort is mainly caused by the multiple execution. The demands on the infrastructure are therefore different from a single large simulation like an aerodynamic simulation with a great number of mesh cells and more complex domain boundaries, where the simulation is distributed over many nodes which must be synchronized after every time step and thus has higher demands on the network connection. A more involved simulation may therefore necessitate the switch to a different infrastructure.
To shield the simulator from the manual adaptation of the provided resources to the different simulation types, we build an infrastructure that helps to provision resources of the correct type and scale transparently.

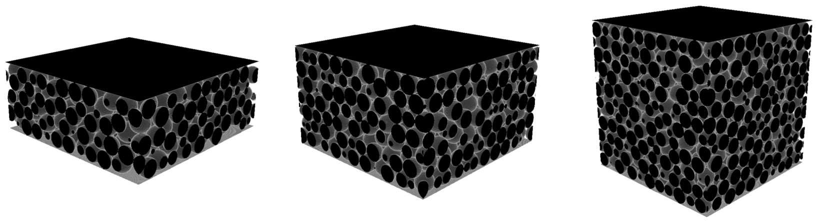

Fig. 1. Exemplary packings of spheres between two plates with a sphere diameter to wall distance ratio of 4, 6 and 10.

3 Model-Driven Resource Provisioning

In our work, we use a formal model of the simulation application topology and its behaviour to automatically provision suitable computing resources. The overall workflow that is implemented by our infrastructure is depicted in Figure 2. To

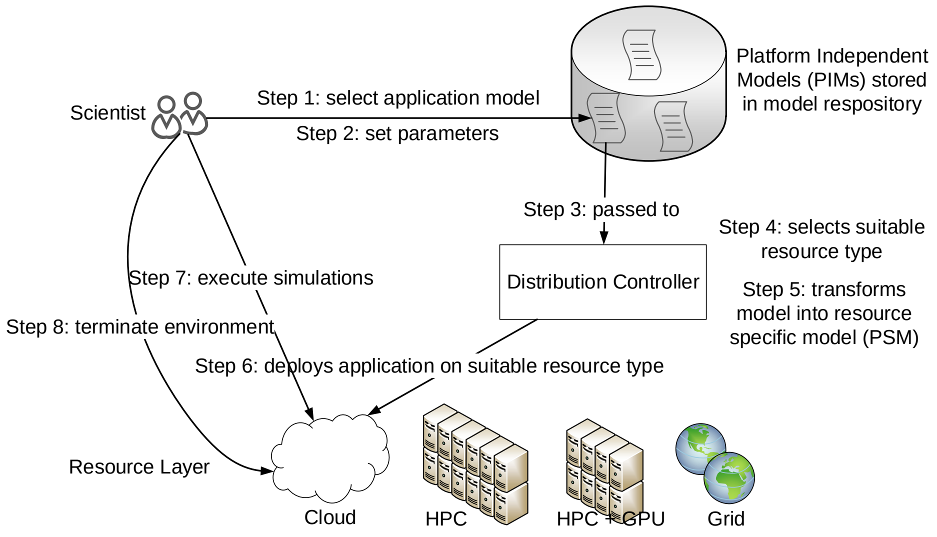
Scientist
Step 1: select application model
Step 2: set parameters
Platform Independent Models (PIMs) stored in model respository
Step 3: passed to
Step 4: selects suitable resource type
Distribution Controller
Step 5: transforms model into resource specific model (PSM)
Step 7: execute simulations
Step 8: terminate environment
Step 6: deploys application on suitable resource type
Resource Layer
Cloud
HPC
HPC + GPU
Grid

Fig. 2. Workflow for model-driven resource provisioning for simulation applications.

distinguish the models used on the different layers we orientate on the notations introduced with the *Model Driven Architecture* (MDA) [15], developed by the *Object Management Group* (OMG).
A *Platform Independent Model* (PIM) encodes the structure and behaviour of the simulation application in a target resource independent way and is stored in a model repository. The simulator is then able to select (Step 1) and adapt (Step 2) the existing models from the repository. The selected and instantiated model is then passed to a *Distribution Controller* (Step 3) that evaluates the parameters of the model and selects the suitable target infrastructure accordingly (Step 4) and finally transforms the selected model into a *Platform Specific Model* (PSM) that matches the requirements of the targeted infrastructure (Step 5). In the next step, the resource provisioning and the automated deployment of the simulation application on the targeted resource is triggered (Step 6). After that, the simulator is given access to the provided resource via a *Command Line Interface* (CLI) and can execute his simulations accordingly (Step 7). When all simulation runs are done, the simulator can collect the output data and triggers the cleanup and termination of the provided infrastructure (Step 8).
Different resource-specific formats for defining the simulation application for the different target infrastructures exist. We adopt the *Topology and Orchestration Specification for Cloud Applications* (TOSCA) [12] which is currently developed by a large technical consortium and allows to define the topology and the behaviour of cloud applications in a provider-agnostic way.
To build a format for describing the PIM, we extend TOSCA to not only be able to capture cloud-specific information, but also the information that is necessary to deploy the application on the other targeted resources, like classical HPC

or HPC+GPU setups. We first concentrate on utilizing the *IBM Load Sharing Facility* (LSF) [9], i.e. generating jobscripts ready for submission. For a given simulation application, this entails taking into account parameters that are fixed upon deployment, for instance the infrastructure model for the compute cluster, as well as parameters that may vary between execution steps like differing mesh resolutions or choices of which data to operate on.
The model-driven approach is also employed to describe the infrastructure on which the simulation application will be deployed and executed. Keeping the application and infrastructure models up to date regarding evolving software versions, for example of the Load Scheduler, is as important as making them available to the simulation scientist in the first place. We therefore intend the repository to be maintained independently for the application and infrastructure models. This way, a PIM of the Load Scheduler is combined with information about a specific cluster setup by its administrator to yield a PSM for the users. The result is combined with the PIM for the simulation application, which can in turn be maintained by its developers. Our approach exemplifies a separation of concerns of the knowledge about the specific compute cluster and about the simulation application from the technical details regarding the transformation of their respective models in the *Distribution Controller*. Consequently, each type of model can be developed by the group of people best equipped to do so, which is a core tenet of MDA.
In summary, the MDA approach addresses the previously stated problems in the following ways:

S1 Separation of concerns is an integral part of MDA - in the case at hand this approach translates to models concerning the simulation application and those describing the various resource types being designed by the application developers/users and compute resource administrators respectively, each contributing their domain-specific knowledge, instead of both sides being handled by the simulation scientist as outlined in (**P1**).

S2 If a convention for describing the interface between applications and resource types is defined and adhered to by the designers of both types of models, the time-investment for making a new resource type elegible for automatic deployment has to be made only once instead of once per application, thereby addressing (**P2**).

S3 Given the model for a specific intended application run and several resource models, programmatically determining the resulting valid combinations can lead to a more well-informed manual decision by the simulation scientist (cf. (**P3**)) and ideally to an automated resource choice.

4 Application Modeling with TOSCA

The *Topology and Orchestration Specification for Cloud Applications* (TOSCA) is a standard which is currently developed by the *Organization for the Advancement of Structured Information Standards* (OASIS). Its goal is to standardize a template format to describe cloud applications in a portable and reusable manner, such that they can be deployed to TOSCA-compliant clouds of different cloud providers. While the targeted resource type for TOSCA are cloud environments, the defined modeling concepts are not resource type specific and we utilize the concepts defined by TOSCA to model simulation applications in a resource type independent way. According to the specification [12], TOSCA is "a language to describe service components and their relationships using a service topology, and it provides for describing the management procedures that create or modify services using orchestration processes." Therefore, it is able to describe both the service structure as well as the processes that can be executed on this structure.

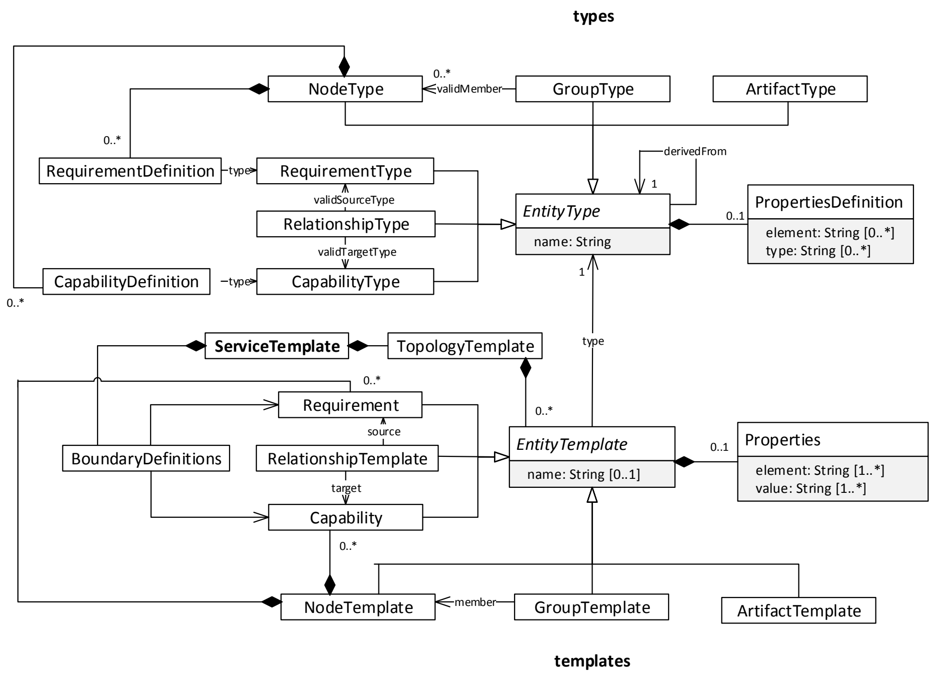

Fig. 3. A simplified subset of the TOSCA metamodel.

A simplified subset of the TOSCA metamodel is depicted in Figure 3. A `ServiceTemplate` captures the structure and also the life cycle operations of the application. For the sake of brevity, we omit the modeling elements that are used to define the life cycle operations, because they are not relevant for the current status of our work. As part of `ServiceTemplate`s, `TopologyTemplate`s can be defined. `TopologyTemplate`s contain `EntityTemplate`s, which can be `NodeTemplate`s that define e.g., the virtual machines or application components, `RelationshipTemplate`s that encode the relationships between the `Node-`

`Templates`, e.g., that a certain application component is deployed on a certain virtual machine, or `GroupTemplates`[4] that allow to group a number of `NodeTemplates`, which e.g., should be scaled together. Additionally, TOSCA defines the `EntityTemplates` `Capability`, `Requirement` and `ArtifactTemplate`. `Capabilities` are used to define that a `NodeTemplate` has a certain ability, e.g., providing a container for running applications, and `Requirements` are used to define that a certain `NodeTemplate` requires a certain `Capability` of another `NodeTemplate`. The aforementioned `RelationshipTemplates` are used to connect the `Requirement` of one `NodeTemplate` with a matching `Capability` of another `NodeTemplate`. The `ArtifactTemplate` is used to model all kinds of artifacts, such as source code, or binaries. All `EntityTemplates` can have `Properties`, e.g., an IP address for a virtual machine, and a certain `type` that references a corresponding `EntityType`. The `EntityType` defines the allowed `Properties` through `PropertyDefintions`, and the allowed `Capabilities` and `Requirements` through `Capability-` and `RequirementDefinitions` respectively. Besides this abstract metamodel, the TOSCA [12] specification defines `normative types` that should be supported by each TOSCA conformant cloud orchestrator. These normative types include e.g., base types for cloud services and virtual machines. More details on the model elements can be found in the TOSCA specifications [12, 13].

5 Evaluation

In the following, we demonstrate how the defined framework can be used to model our use case application from fluid mechanics and provision resources on two different target platforms, namely an *High Performance Computing* (HPC) system and an *Infrastructure as a Service* (IaaS) cloud. Therefore, we present and discuss the models in the provisioning process for the different target resources. To foster comprehensibility, we slightly modified the models and omit some technical details.

5.1 Utilized Tooling

The *Eclipse Modeling Framework* (EMF) defines a common standard for structured data models, which conform to metamodels specified in the *Ecore* format [19]. We automatically converted the *XML Schema Definition* (XSD)-based TOSCA specification to such a metamodel using the EMF tools, and parsed the normative type definitions specified in *YAML Aint Markup Language* (YAML) manually. Once the complete TOSCA metamodel was available in the *Ecore* format, we were able to edit the application and resource models, verify them against the constraints imposed by the respective metamodel, and persist them

[4] At the time of this writing, two versions of the TOSCA standard exist, a formalized version [12] in XML and a simplified rendering in YAML [13]. `GroupTemplates` and `GroupTypes` are currently part of the TOSCA YAML rendering, but not part of the TOSCA XML specification.

using the *XML Metadata Interchange* (XMI) format.
With the help of the domain-specific languages provided by the *Eclipse Epsilon* project [20], *Model-to-Model* (M2M) and *Model-to-Text* (M2T) transformations can be performed using various modeling techniques, in particular EMF. For example, we implemented M2M transformations with the *Epsilon Transformation Language* (ETL) in order to obtain the PSM for a concrete setup from the PIM of the respective resource type, modeling an IaaS cloud standard or an HPC batch system. Using the resulting model, the application PIM is transformed to the PSM for deployment in a similar fashion. M2T transformations, on the other hand, are realized using the *Epsilon Generation Language* (EGL), a template-based format that is suitable for amend generic IaaS deployment scripts or HPC jobscripts with the data of the deployment model.
While the metamodels, models and transformations can be implemented and debugged in the Eclipse workbench for development purposes, in order to enable the user to focus on application deployment, we implement a command-line interface which handles model transformations and template rendering as a standalone application.

5.2 Model of the fluid mechanics application

In Figure 4 the application model for *Adaptive Mesh Refinement in Object-oriented C++* (AMROC) [4], the fluid solver framework our simulation use case is based on, is depicted. The software uses the parallelization standard *Message Passing Interface* (MPI).

Type definitions: The central node type `AMROCNType` is equipped with three requirement definitions: `MPIClusterRDef` models the fact that an MPI cluster is necessary to run the software and allows specifying the necessary number of nodes and the required MPI version via the properties of the corresponding capability type `MPIClusterCType`, the requirement definition `PackageRDef` defines the requirement for a specific mpi implementation given by the respective capability type `PackageCType`, and finally `HostRDef` addresses the hardware requirements imposed on the involved compute nodes, namely the amount of memory and CPU cores, which constitute the properties of its respective capability type `HostCType`. In addition, an artifact type `AnsibleRoleAType` enables the application model to provide scripts for the configuration management tool Ansible[5].

Templates: The requirements of a concrete simulation application based on AMROC are modeled by a corresponding node template `AMROCNTemplate` to which instances of the requirement definitions are assigned, providing values for the properties specified by `MPIClusterCType`, `PackageCType`, and `HostCType`. Finally, `AMROCCode` references the provided configuration management script.

[5] https://www.ansible.com/

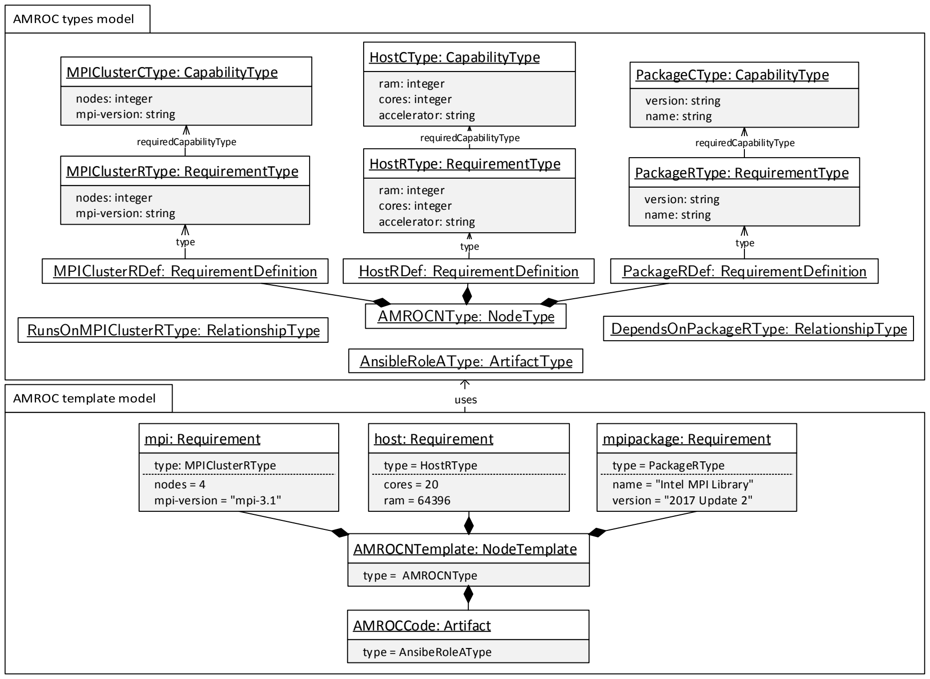

Fig. 4. Platform independent model of the fluid mechanics application.

5.3 Model of batch target system

Our model of an HPC cluster using the *Load Sharing Facility* (LSF) as the batch system is shown in Figure 5. Again, the graphical representation is separated for type and template definitions: While the types section is specific only to the batch system, the templates represent parts of the Scientific Compute Cluster configuration as it is hosted at the *Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen* (GWDG).

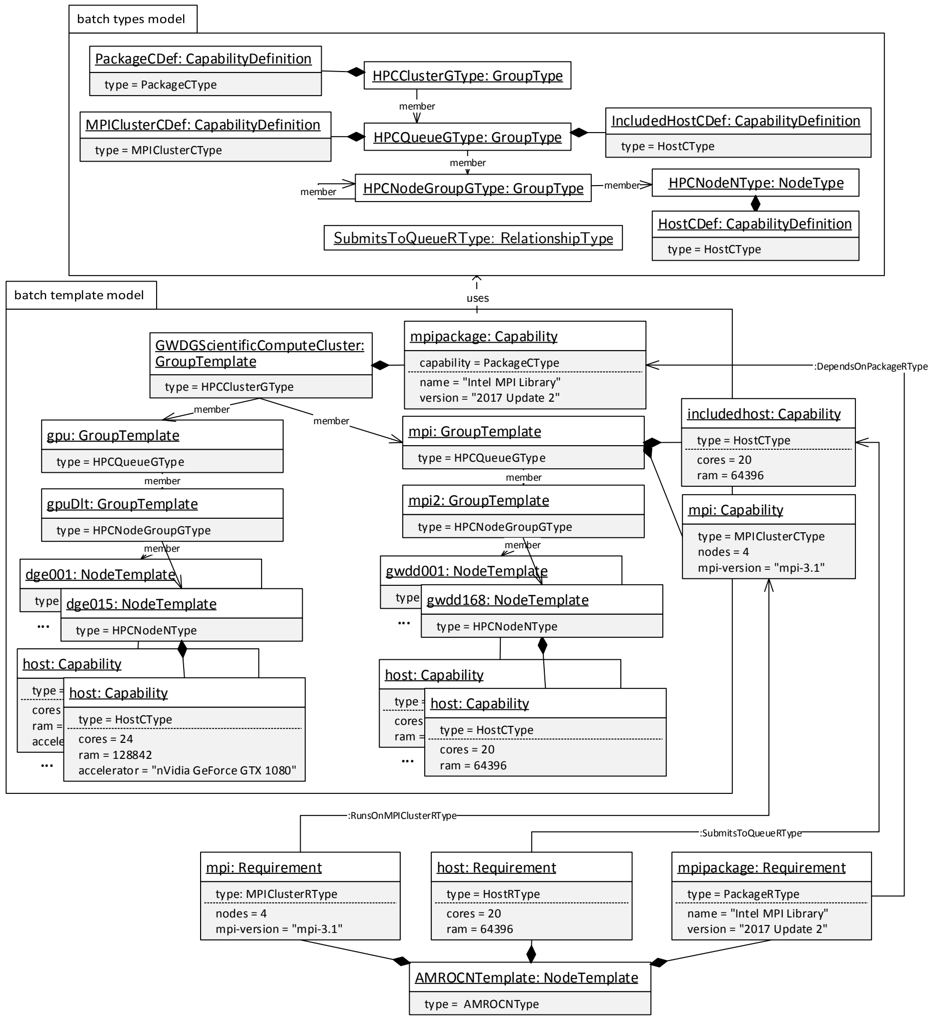

Fig. 5. Provisioning and deployment model for the HPC target resource type.

Type definitions (LSF cluster): The group type `HPCClusterGType` serves as a root element providing the capability `PackageCDef`, which exposes preinstalled software packages, usually made available in the form of environment modules. These are specified using the properties of the corresponding capability type `PackageCType`. The compute nodes in an LSF cluster, which are modeled by `HPCNodeNType`, are organized in a tree-like structure of host groups modeled by group elements `HPCNodeGType`. A capability definition `HostCDef` of type `HostCType` (cf. Figure 4) is used to define each node's hardware specifications.
For the available job submission queues of type `HPCQueueGType` further capabilities are generated dynamically: `IncludedHostCDef`, which is of the same capability type used for individual hosts, allows the Distribution Controller to match the application model's requirements for `HostCType` against the configured queues directly. `MPIClusterCDef` marks the queue as suitable for submitting MPI jobs (cf. Figure 4 for the definition of `MPIClusterCType`).
The relationship type `SubmitsToQueueRType` is used for choosing one of the queues during deployment of the application. Finally, the queue references one or more host groups as members.

Templates (GWDG Scientific Compute Cluster): The central group type `HPC-ClusterGType` is instantiated as the group template `GWDGScientificCompute-Cluster`. The capability type `PackageCType` is used to specify the available MPI library version by its instance `mpipackage`.
Two representative examples, the `mpi` and `gpu` queues are given as group instances of `HPCQueueGType`, along with representatively chosen host groups as members of type `HPCNodeGType`. Their members are in turn models of the available compute nodes, specified by a node template of type `HPCNodeNType` each.
The hardware capabilities of these nodes, namely the number of cores, amount of installed memory and type of eventually installed GPU accelerator are described by capability assignments of type `HostCType`.

AMROC deployment on the batch system: As an example, the model elements relevant for deploying the AMROC application and submitting a job to the mpi queue are shown as well: An instance `AMROCNTemplate` of `AMROCNType` provides concrete values for its associated requirements, which are matched to the globally defined `mpipackage` as well as the queue-specific capabilities `includedhost` and `mpi` via the respectively suitable relationship.

5.4 Model of IaaS cloud target system

Figure 6 depicts the model for the IaaS cloud system. One fundamental difference to the batch system is that compute nodes can be created and configured on demand according to simulation application requirements, including the (virtual) hardware, the installed software and the operating system on the compute nodes.

Type definitions (Cloud orchestration): The type definitions depicted in the upper part of the figure orientate on the TOSCA types that are defined and utilized

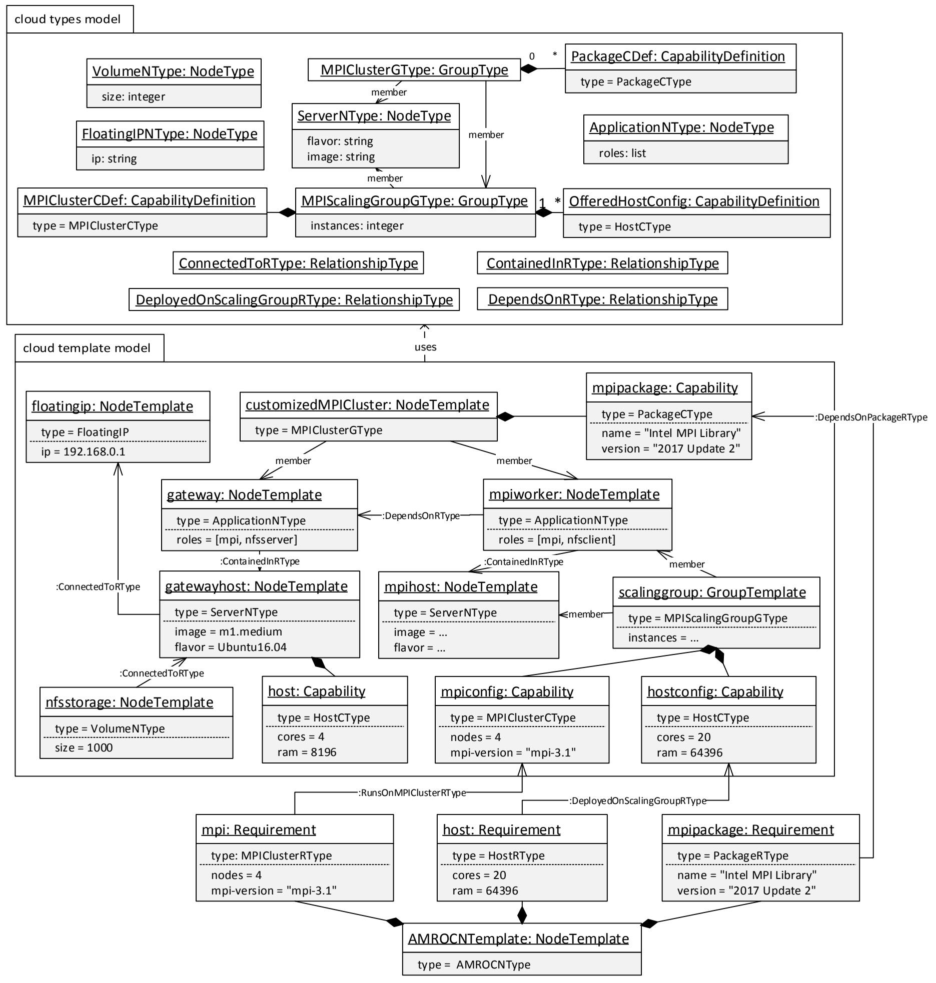

Fig. 6. Provisioning and deployment for the IaaS target resource type.

by the TOSCA-compliant cloud orchestrator Cloudify[6] for modeling cloud resources of an OpenStack[7] cloud. The group type `MPIClusterGType` serves as a root element providing the capability `PackageCDef`, which exposes software packages for which installation scripts are available and which can therefore be installed on demand. The node type `ServerNType` abstracts the notion of a *Virtual Machine* (VM) that can be started in the IaaS cloud. Its properties `flavor` and `image` encode the hardware configuration and the operating system respectively. The node type `VolumeNType` models an external block-storage device that can be attached to a running VM and the node type `FloatingIPNType` allows to define a publicly reachable IP address for a VM. Finally, `ApplicationNType` is a node type used to model an application that should be installed on the deployed virtual machine, in the case of the prototypical implementation via Ansible roles referenced by its property `roles`.
The group type `ScalingGroupGType` allows to group templates of the type `ServerNType` and `ApplicationNType` to be scaled together. Moreover, it is used to define the virtual hardware configuration of the contained VMs via its capability definition `OfferedHostConfig` of type `HostCType`. The fact that the group is configured as an MPI cluster is captured by defining the capability `MPIClusterCDef` (cf. Figure 4 for the definitions of both capability types). This capability is also the target of the relationship `DeployedOnScalingGroup` used for application deployment.
Three additional relationship types `ConnectedToRType`, `ContainedInRType` and `DependsOnRType` are defined that have the following semantics: `ConnectedToRType` expresses a connection between two entity types, `ContainedInRType` expresses the fact that an entitiy type is part of another entity type and the relationship type `DependsOnRType` expresses a general dependency between two entity types.

Templates (MPI cluster infrastructure): The group type `MPIClusterGType` is instantiated as the node template `customizedMPICluster`, whose `PackageCType` instance `mpipackage` specifies the MPI version that can be installed.
The node template `gatewayhost` models a VM that is reachable from outside the cloud via an assigned IP-address, modeled by the node template `floatingip`, while the node template `gateway` models its software configuration. The `gatewayhost` is connected to an *Network File System* (NFS)-storage volume, modeled by the node template `nfsstorage` that is shared among the MPI-enabled worker nodes. The MPI worker nodes are modeled by the node template `mpihost` and its assigned software configurations `mpiworker`. The node templates `mpihost` and `mpiworker` are members of a group template `scalinggroup`: According to the number of compute nodes that are required by the simulation application (cf. Figure 4)), the property `instances` is used to replicate the node templates given by the property `members` and all relationships connecting them. In this way, we are able to scale the number of compute nodes according to the resource

[6] http://cloudify.co
[7] http://www.openstack.org

demand of the application. The capabilities `mpiconfig` and `hostconfig` assignet to `scalinggroup` provide concrete values for the available MPI version and the virtual hardware specifications respectively.

AMROC deployment in the cloud: As in the aforementioned case of the batch system, the deployment model adds the instance `AMROCNTemplate` of `AMROCNType` providing concrete values for the associated hardware and MPI requirements which are connected to the infrastructure model's capabilities by relationships.

6 Discussion

Our approach applies the MDA solution strategies to the problems outlined above as follows:

(P1,S1) We were able to construct the application model for AMROC in a resource-independent way, while on the other hand the models describing an IaaS cloud and an HPC cluster using LSF are focused on the logic of the respective resource types only. These examples indicate that modeling anything but the application itself can in principle be removed from the scope of the simulation scientist.

(P2,S2) Exposing both the requirements of the simulation application and the capabilities of both resource types considered here by referring to a common set of capability types makes application and resource models respectively interchangeable.

(P3,S3) The existence of a particular Capability within a given resource model as well as the concrete values of its properties enable an automatic choice of a suitable compute resource.

6.1 Limitations

Currently, we focus on the provisioning and deployment of the resources for the simulation application, and especially do not consider the following points:

- We assume the compiled simulation to be able to run on all eligible resource types. However, incorporating the build process as part of the model transformation would enable the same application to be automatically deployed to all hardware architectures it compiled for.
- Platform-dependent code modifications are currently out of scope, therefore only the simulation code as whole is part of the application model. Examples would be the configuration of load-balancing schemes in a parallelization workload, moving parts of a calculation to an accelerator depending on its availability or even generating the simulation code from a model of the underlying algorithm (cf. [14]).
- In the present work, the automated choice of a compute resource is based solely on the data found in a corresponding resource model. Including the results from monitoring previous jobs, in particular in the form of job chains composed of comparable workloads such as parameter studies, would enable us to adjust the parameters of subsequent deployments more precisely.

- The scope of our architecture is currently restricted to handling a single application run. In practice, a combination of multiple jobs, such as the cleaning of input data, choosing the next simulation step based on the data of the previous one and reduction of the results, compose the entire workflow that is ultimately of interest. A model of this high-level view of the simulation needs to include the results of the present work as a combinable unit. Vukojevic-Haupt et al. [22] describe an approach for the provisioning of a cloud-based middleware intended to particularly handle simulation workflows.
- We do not consider data management tasks associated with the application run at this point, such as copying data back and forth between the simulation scientist's system and the compute resource or persisting and sharing the results using independent research data management infrastructures.

7 Related Work

Different projects address specifically the use of models or *Domain-Specific Languages* (DSLs) to target the execution of scientific software in a cloud environment, e.g., Bunch et al. [3] define a DSL for the management of HPC applications in the cloud, and Qashsa et al. [16] utilize TOSCA to model scientific workflows for the cloud. Furthermore research has been conducted to provide scientific applications as services in a cloud environment, e.g., Vukojevic-Haupt et al. [21] define a middleware to deploy scientific applications as services, and Limmer et al. [10] utilize the cloud-standard *Open Cloud Computing Interface* (OCCI) [11] to steer simulation applications in the cloud. Similar to the problem of heterogeneous resource types, is the problem of vendor-specific *Application Programming Interfaces* (APIs) and the service heterogeneity of different cloud providers, because this makes it infeasible to switch between the offerings. Ardagna et al. [1] propose the use of MDA to model cloud applications in a cloud-provider independent fashion, and Quinton et al. [17] provide a platform to select and configure a specific cloud-offering based on models. Further approaches aiming to provide provider-agnostic frameworks for cloud-based applications [5, 7] and to improve cloud interoperability by combining existing applications using TOSCA [18] exist, but none of the aforementioned solutions discuss the problem of addressing different types of target resources outside the realm of cloud providers. The OCCIware project [23] showed that also the cloud standard OCCI can be used to model and manage all kinds of resources, but the standard does not provide the functionality to define requirements and capabilities as needed for our architecture, and the automated mapping to a specific infrastructure type is not addressed.

Flissi et al. [6] developed a model-driven method to deploy distributed systems on Grids. In contrast to our work, they do not consider the automated mapping to a specific resource and also do not face the problem with dynamically adaptable resource target types, such as cloud systems. Ober et al. [14] discuss the use of model-driven engineering techniques for the development of HPC applications and Arkin et al. [2] provide a model-driven method to map algorithms to a certain parallel computing platforms, such as MPI or OpenMP. In contrast to our work, both works directly consider the developed code, whereby we focus on the provisioning of the computing resources.

8 Conclusions and Outlook

We develop an architecture which provides a transparent resource provisioning mechanism for simulation applications for heterogeneous computing infrastructures, which are today's reality and in many data centers. The goal is to shield the simulation scientist from complicated infrastructure internals. In this paper, we presented the initial architecture, which orientates on the MDA and demonstrated its feasibility with the help of a LBM simulation from fluid mechanics. Future work includes the adaptation of the strict resource requirements modeled on a per-host basis in our approach to a more flexible format that allows the Distribution Controller to split, for example, the same total amount of CPU cores and memory in different ways according to the available hardware. Another possible extension of the architecture consists of logging and analyzing the achieved application performance in order to improve the choices made in following iterations of deploying the same application.

Acknowledgements

We thank the Simulationswissenschaftliches Zentrum Clausthal-Göttingen (SWZ) for financial support.

References

1. Ardagna, D., Nitto, E.D., Mohagheghi, P., Mosser, S., Ballagny, C., D'Andria, F., Casale, G., Matthews, P., Nechifor, C.S., Petcu, D., Gericke, A., Sheridan, C.: Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. In: 2012 4th International Workshop on Modeling in Software Engineering (MISE). pp. 50–56. IEEE (June 2012). https://doi.org/10.1109/MISE.2012.6226014
2. Arkın, E., Tekinerdogan, B., İmre, K.M.: Model-driven approach for supporting the mapping of parallel algorithms to parallel computing platforms. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P. (eds.) Model-Driven Engineering Languages and Systems. pp. 757–773. Springer (2013). https://doi.org/10.1007/978-3-642-41533-3_46
3. Bunch, C., Chohan, N., Krintz, C., Shams, K.: Neptune: A domain specific language for deploying hpc software on cloud platforms. In: Proceedings of the 2nd International Workshop on Scientific Cloud Computing. pp. 59–68. ScienceCloud '11, ACM (2011). https://doi.org/10.1145/1996109.1996120
4. Deiterding, R.: AMROC - Adaptive Mesh Refinement in Object-oriented C++ (2017), Available Online: http://www.vtf.website/asc/wiki/bin/view/Amroc/WebHome, Last Retrieved: 08.11.2017

5. Di Martino, B., Petcu, D., Cossu, R., Goncalves, P., Máhr, T., Loichate, M.: Building a mosaic of clouds. In: Guarracino, M.R., Vivien, F., Träff, J.L., Cannatoro, M., Danelutto, M., Hast, A., Perla, F., Knüpfer, A., Di Martino, B., Alexander, M. (eds.) Euro-Par 2010 Parallel Processing Workshops. pp. 571–578. Springer (2011). https://doi.org/10.1007/978-3-642-21878-1_70
6. Flissi, A., Dubus, J., Dolet, N., Merle, P.: Deploying on the grid with deployware. In: 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID). pp. 177–184. IEEE (2008). https://doi.org/10.1109/CCGRID.2008.59
7. Guilln, J., Miranda, J., Murillo, J.M., Canal, C.: A service-oriented framework for developing cross cloud migratable software. Journal of Systems and Software **86**(9), 2294 – 2308 (2013). https://doi.org/10.1016/j.jss.2012.12.033
8. Hofmann, S., Bufe, A., Brenner, G., Turek, T.: Pressure drop study on packings of differently shaped particles in milli-structured channels. Chemical Engineering Science **155**, 376–385 (2016). https://doi.org/10.1016/j.ces.2016.08.011
9. IBM Corporation: Introduction to IBM Platform LSF, Available Online: https://www.ibm.com/support/knowledgecenter/SSETD4_9.1.2/lsf_foundations/lsf_introduction_to.html, Last Retrieved: 08.11.2017
10. Limmer, S., Srba, M., Fey, D.: Performance investigation and tuning in the interoperable cloud4e platform. In: Euro-Par 2014: Parallel Processing Workshops. pp. 85–96. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14313-2_8
11. Nyrén, R., Edmonds, A., Papaspyrou, A., Metsch, T., Parák, B.: Open Cloud Computing Interface - Core (September 2016), [Available Online: http://ogf.org/documents/GFD.221.pdf]
12. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) 1.0 (November 2013), Available Online: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html, Last Retrieved: 08.11.2017
13. OASIS: TOSCA Simple Profile in YAML Version 1.1 (August 2016), Available Online: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.1/csprd01/TOSCA-Simple-Profile-YAML-v1.1-csprd01.html, Last Retrieved: 08.11.2017
14. Ober, I., Palyart, M., Bruel, J.M., Lugato, D.: On the use of models for high-performance scientific computing applications: an experience report. Software & Systems Modeling **17**(1), 319–342 (Feb 2018). https://doi.org/10.1007/s10270-016-0518-0
15. Object Management Group: Model Driven Architecture, Available Online: http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf, Last Retrieved: 08.11.2017
16. Qasha, R., Cala, J., Watson, P.: Towards automated workflow deployment in the cloud using tosca. In: 2015 IEEE 8th International Conference on Cloud Computing. pp. 1037–1040. IEEE (2015). https://doi.org/10.1109/CLOUD.2015.146
17. Quinton, C., Romero, D., Duchien, L.: Saloon: a platform for selecting and configuring cloud environments. Software: Practice and Experience **46**(1), 55–78 (2016). https://doi.org/10.1002/spe.2311
18. Soldani, J., Binz, T., Breitenbcher, U., Leymann, F., Brogi, A.: Toscamart: A method for adapting and reusing cloud applications. Journal of Systems and Software **113**, 395 – 406 (2016). https://doi.org/10.1016/j.jss.2015.12.025
19. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0. Addison-Wesley Professional, 2nd edn. (2009)
20. The Eclipse Foundation: Epsilon, Available Online: https://eclipse.org/epsilon/, Last Retrieved: 08.11.2017

21. Vukojevic-Haupt, K., Haupt, F., Leymann, F.: On-demand provisioning of workflow middleware and services into the cloud: an overview. Computing **99**(2), 147–162 (Feb 2017). https://doi.org/10.1007/s00607-016-0521-x
22. Vukojevic-Haupt, K., Haupt, F., Leymann, F., Reinfurt, L.: Bootstrapping complex workflow middleware systems into the cloud. In: 2015 IEEE 11th International Conference on e-Science. pp. 126–135. IEEE (2015). https://doi.org/10.1109/eScience.2015.69
23. Zalila, F., Challita, S., Merle, P.: A model-driven tool chain for occi. In: Panetto, H., Debruyne, C., Gaaloul, W., Papazoglou, M., Paschke, A., Ardagna, C.A., Meersman, R. (eds.) On the Move to Meaningful Internet Systems. OTM 2017 Conferences. pp. 389–409. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69462-7_26