Towards a Unified Requirements Model for Distributed High Performance Computing

Michał Śmiałek, Kamil Rybiński, Radosław Roszczyk and Krzysztof Marek

Abstract High Performance Computing (HPC) consists in development and execution of sophisticated computation applications, developed by highly skilled IT personnel. Several past studies report significant problems with applying HPC in industry practice. This is caused by lack of necessary IT skills in developing highly parallelised and distributed computation software. This calls for new methods to reduce software development effort when constructing new computation applications. In this paper we propose a generic requirements model consisting of a conceptual domain specification, unified domain vocabulary and use-case-based functional requirements. Vocabulary definition provides detailed clarifications of HPC fundamental component elements and their role in the system. Further we address security issues by providing transparency principles for HPC. We also propose a research agenda that leads to the creation of a model-based software development system dedicated to building Distributed HPC applications at a high level of abstraction, with the object of making HPC more available for smaller institutions.

1 Introduction and motivation

One of the critical problems in technology innovation activities is the efficient processing of large amounts of data using sophisticated computation algorithms. Computations like this need highly efficient supercomputing systems which are expensive, complex to use and can be configured and programmed only by highly skilled IT personnel. Such requirements pose a big problem for smaller groups like small and medium companies (SMEs), innovative start-ups or small research institutions. These organisations are unable to make a significant financial commitment. Despite having necessary domain knowledge (e.g. solving engineering problems), due to their lack of software development competencies, they are unable to benefit from HPC.

Warsaw University of Technology, e-mail: michal.smialek@ee.pw.edu.pl

Furthermore, High-Performance Computing (HPC) [12] is also in the process of dynamic changes. This change is associated with the emergence of Cloud Computing and its application to HPC [29]. This new trend can be compared to older paradigms like Grid Computing, and generally – to distributed systems [9]. While growth in hardware and low-level operating system capabilities is tremendous, the capabilities to develop HPC software seem to lag behind. As Giles and Reguly [10] point out, this is caused by quite slow adaptation of existing code libraries to new architectures and parallel distributed processing. Moreover, orchestration of computations in a distributed, cloud-enabled environment poses significant problems to programmers [25]. Considering this, Van De Vanter et al. [28] observe the need for new software development infrastructures, explicitly dedicated to HPC. An interesting study by Schmidberger and Brugge [26] shows that understanding of software engineering methods among HPC application developers is not yet satisfactory. What is more, these methods need to be significantly adapted to the specificity of scientific and engineering computations. An example approach to this issue is presented by Li et al. [18, 17]. It proposes a lightweight framework to define requirements for computation applications using a domain-specific approach [5].

Currently, many companies, who are willing to take a leap into HPC solutions, are forced to use outside organizations to rewrite their original programs. For them, it is the only way to make their solution parallelised and optimised. This process often takes months or even years, because of a lack of domain knowledge among HPC specialists that rewrite code. A good example is the SHAPE project, whose main goal is to provide SMEs with both HPC experts and computation power. Despite such huge help, the results are only able to work on specific architectures and require further development [27]. Moreover, in many cases, the projects were discontinued after the end of SHAPE support. They often did not translate into broader use of HPC in these SMEs. This shows how difficult it is for smaller organisations to use HPC, despite significant benefits.

Therefore we can argue that raising the level of abstraction when programming an HPC application can significantly reduce complexity of programs and increase programmer productivity. In more traditional approaches, this leads to constructing code libraries optimised for HPC, compatible with more modern – object-oriented – programming languages (e.g. Java as compared to FORTRAN or C) [22]. Such approaches allow for better modularisation of code which becomes more elegant, easy to understand and portable. However, they do not remove technological complexity associated with programming platforms and constructs of any contemporary programming language. Even the solutions whose main purpose was to enable easy communication management in object-oriented programming languages, turn out to require complex knowledge of HPC. For example, OpusJava [15, 16] was designed to enable high performance computing in large distributed systems by hiding many low-level details, including thread synchronization and communication. As a result, it made parallel programming easier for IT specialists. Unfortunately it didn't remove the need for sophisticated HPC knowledge and still necessitates significant programming skills. We argue that in order to cope with such complexity we should use model-driven techniques.

Considerations on model-driven approaches lead to the construction of certain domain-specific languages oriented specifically on developing HPC applications. An important problem when developing such languages is to understand the sophisticated nature of large-scale scientific and engineering computations. This is reflected by the complexity and variety of computation problems, structure of the computation algorithms and distribution of computation tasks, as pointed out in past research [2, 23, 6]. Moreover, it can be noted that the terminology for HPC is not standardised. For instance, Dongarra et al. argue about the meaning of certain terms related to clustering and parallelisation [8].

In this position paper we propose an initial conceptual model, functional scope and research agenda for creating and validating a Distributed HPC (DHPC) software development system that fulfils the above general requirements. The goal is to offer a coherent holistic solution: an easy-to-program and affordable supercomputing platform that could improve innovative potential and reduce time-to-market for high-technology services and products. At the same time, it would allow to develop a gradually developing network of independent computation resources that would follow the same underlying computation model and language.

2 Related work

The idea to use high level of abstraction as a solution to the problem of accessibility to HPC by SMEs and small research institutions, is not new. The Legion system [11] is one of the first attempts to provide functional requirements for distributed HPC. In this work, DHPC is referred-to as the Worldwide Virtual Computer. Much have changed since that moment but the need for unified functional requirements remains. More recently, such requirements can be expressed using domain specific languages. For instance, the Neptune system [6] formulates a formally defined language for specifying the distribution aspects for HPC applications. Other approaches concentrate on specific application domains. For instance, ExaSlang [**?**] was developed to specify numerical solvers based on the multigrid method. Another example is the system developed by Hernandez et al. [13] to automate the development of grid-based applications. However, we can argue that an ultimate solution should be independent of a specific problem domain or class of computation problems. Such a goal was the basis for constructing HPCML (HPC Modelling Language) [24]. The language uses graphical notation to denote flow of computation tasks, parallelism of tasks and data, and other issues. However, it concentrates only on the computation aspect, and does not describe variuos other aspects necessary to organise the whole DHPC environment.

As far as model transformation techniques are concerned, Palyart et al. propose an interesting method called MDE4HPC [23] that is based on developing higher-level computation models that can be transformed to code. These models abstract away all the platform-specific issues and concentrate on the essence – the computation problem. Another approach is proposed by Membarth et al. [20]. It involves a domain-

specific language which is used to develop computation models. The underlying transformation engine can generate code for various processing environments (CPUs and GPUs) thus handling the important problem of heterogeneity in HPC.

Analysis of existing literature shows very limited research that includes approaches to formulate conceptual models and functional requirements for DHPC systems. To our best knowledge, supported by queries to ACM DL, IEEE Xplore, Scopus and Google Scholar, there are no direct solutions of this kind. Analysis of literature shows that research focuses mostly on solving specific problems associated with HPC. Authors do not specify any functional requirements for their solutions, nor any conceptual models. They provide non-functional requirements (eg. performance parameters), describe their technical solution and assume implementation of certain user expectations that are not explicitly formulated as functional requirements. A good example of such practice is the XtremWeb project [7]. Its goal was to create a peer-to-peer computation system, that would enable parallel computing in a distributed large-scale system. Available literature on XtremWeb focuses on the technical solution, system capabilities and security aspects. User requirements are briefly mentioned, and no conceptual model is given. In our opinion, one of the reasons behind lack of universality in current solutions is different understanding of user expectations. Therefore unifying the domain model and functional requirements for DHPC systems, could ensure broader and more universal solutions in the future.

If we summarise the above considerations we can note that the ultimate goal is to create an integrated general-purpose software development system dedicated to HPC applications. Such system should allow for developing high-level computation models through a domain-specific language that abstracts away the platform-specific issues and concentrates on the essence – defining the computation itself. At the same time, the system should be able to translate this DSL-defined computation into a well-architected computation application. This application should follow best design practices for HPC [1] and be able to quickly react to changing requirements [19]. Finally, it should scale to cover variable models of computing resource utilisation [29].

In addition to many technical aspects that should be taken into account to build a distributed computing system, security of the solution should be taken into account. One of the most important aspects is trust management in a distributed system [3]. The security of a distributed system is generally more complex than the security of an independent system. Current computer security concepts assume that trust is assigned to an element of a distributed system based on a point of view. This security mechanism for distributed file systems solves many performance and security problems in current systems [4] [14].

3 Domain model for distributed HPC

In order to obtain a unified domain model for HPC we consider four fundamental elements of any computation problem: the Computation Application, the Computation

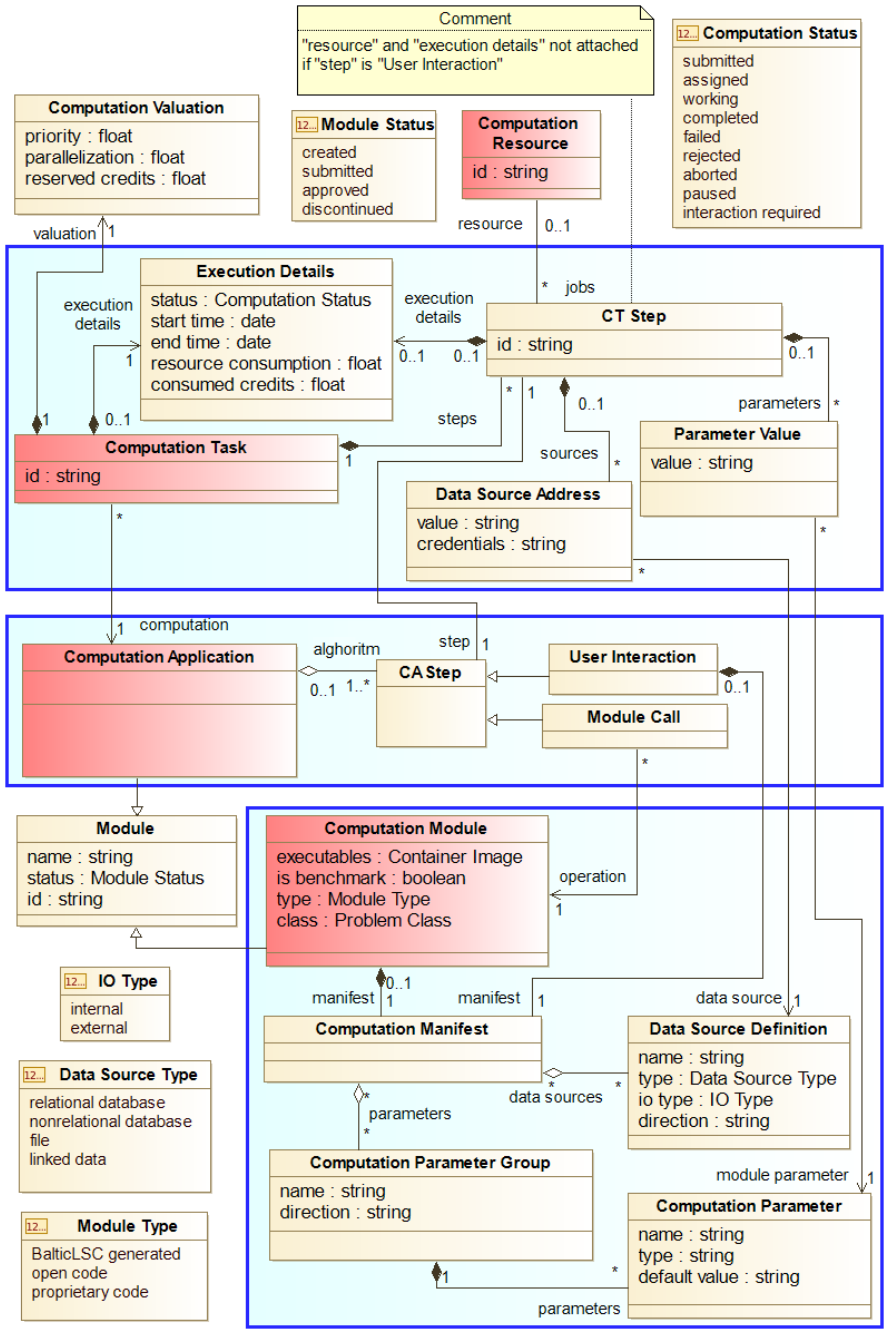

Fig. 1 Domain model for Distributed HPC computations

Task, the Computation Module and the Computation Resource. These four elements are highlighted in Figure 1. Generally, HPC computations are defined through Computation Applications (CA) that constitute high-level computation "programs". These programs can be provided by end users of a computational platform or by advanced developers involved in the creation of such applications. The CAs are composed of calls to self-contained Computation Modules (CM). Each CM performs a well-defined, atomic piece of calculations. Calculations at this level can be performed independently of each other in a parallel or a sequential way. Execution (instantiation) of a CA and the contained CMs is performed through creating a Computation Task (CT). Each task can be executed on several Computation Resources (CR), i.e. on specific machines (computer hardware with OS) equipped with specialized computing nodes (virtual machines or docker containers).

Each Computation Application represents a sequence of CA Steps (see again Figure 1) needed to perform the given computation (the algorithm). Within the algorithm, we can distinguish two types of calculation steps. The first type (User Interaction) carries out all kinds of interactions with the users. The second type (Module Call) is responsible for calling specific Computation Modules and controlling of data flow between the modules. These two types of steps are ordered in sequences, which compose into a specific calculation algorithm that realizes a specific end user request. Depending on the construction of the algorithm, some steps may need to be performed in parallel, while others must be performed sequentially. When computations are performed in parallel, a significant issue is to synchronise data produced by the modules working in parallel. This needs to be done before going to the next steps that normally would require data produced by the previous parallel modules.

It can be noted that the conceptual model in Figure 1 does not present any details regarding the flow of CA Steps. This should be part of a detailed meta-model for a new Computation Application Language which is out of scope of this paper. This language would define constructs for representing flow of computations and data in a distributed computing environment, such as:

- algorithm branching for parallel execution,
- synchronization of data from parallel steps,
- retrying tasks in case of failure (in accordance with the defined retry policy),
- data transfer between computation modules,
- parallelization of computation modules,
- data exchange between modules,
- execution of computation modules.

It would also include constructs to define details of user interactions, including such operations as:

- entering parameters for the algorithms contained in the computation modules,
- determining input sources for computation modules,
- determining pointers to storage for the calculation results,
- entering parameters that control computations (performance, security etc.).

Let us now consider the role of Computation Modules. Their primary function is to store and the executable code for a given computational component in the form of a Container Image. The size of this image depends on the actual computation algorithm. The module contents can range from simple algebraic operations to complex algorithms that implement AI tasks. For every CM we have to determine its type and class. The Module Type determines the level of access to the executable module code, and thus determines the level of trust during its execution. This mechanism has been introduced to secure the computing platform against unauthorized use. It also protects the interests of participants offering their machines for computational purposes. The Problem Class applies to the mechanisms associated with benchmarking of COmputation Resources, as explained further in this section.

Each Computation Module, apart from the executable code, is equipped with a Computation Manifest. It aggregates information about data sources (Data Source Definition) and possible computation parameters of the calculation module itself (like computation performance) and input parameters for the computation algorithm. This information thus provides complete information about the input and the output for a given module and assures a mechanism of cooperation between modules. In addition, this manifest contains information on the possibility of parallelisation of the computation task and requirements for possible synchronization between the modules.

The underlying general assumption is that each CM processes some data on its input and generates some data on its output. The computations are controlled by the module parameters (e.g. maximum number of iterations, calculation threshold, cooperation with the environment, data exchange, and other parameters). We should also note the assumption that data is normally not transferred directly to the CM's storage space. Instead, each CM accesses the data through a specific separator (proxy) that gives access to the external or internal storage of the data.

Each execution of a Computation Application with the related CMs, creates a new instance of Computation Task. This task is described by so-called execution details (see the Execution Details class in Figure 1). They contain performance parameters, such as the consumption of computing resources in the form of CPU usage time, number of cores, memory occupancy, disk space usage, the amount of data transmitted by the network and other parameters of the execution environment. Before starting any computation task, the end-user must declare an estimated target resource consumption for a given CT. This initial estimate can be based on the requirements of the computational algorithm (provided by the algorithm developer) and the amount of data that will be processed. At the stage of task execution planning, the system will take into account the parameters declared by the user and – based on them – appropriately arrange the CT task to be carried out.

When creating a performance plan for a CT, the computation runtime system creates an appropriate CT Step for each CA Step contained in the CA's algorithm. A unique status is created for each CT Step (Computation Status). The mechanism for collecting and supervising statuses allows for ongoing analysis of the progress of calculations in a distributed environment. In addition, each CT Step stores specific values for its data sources and the current module execution parameters. The actual

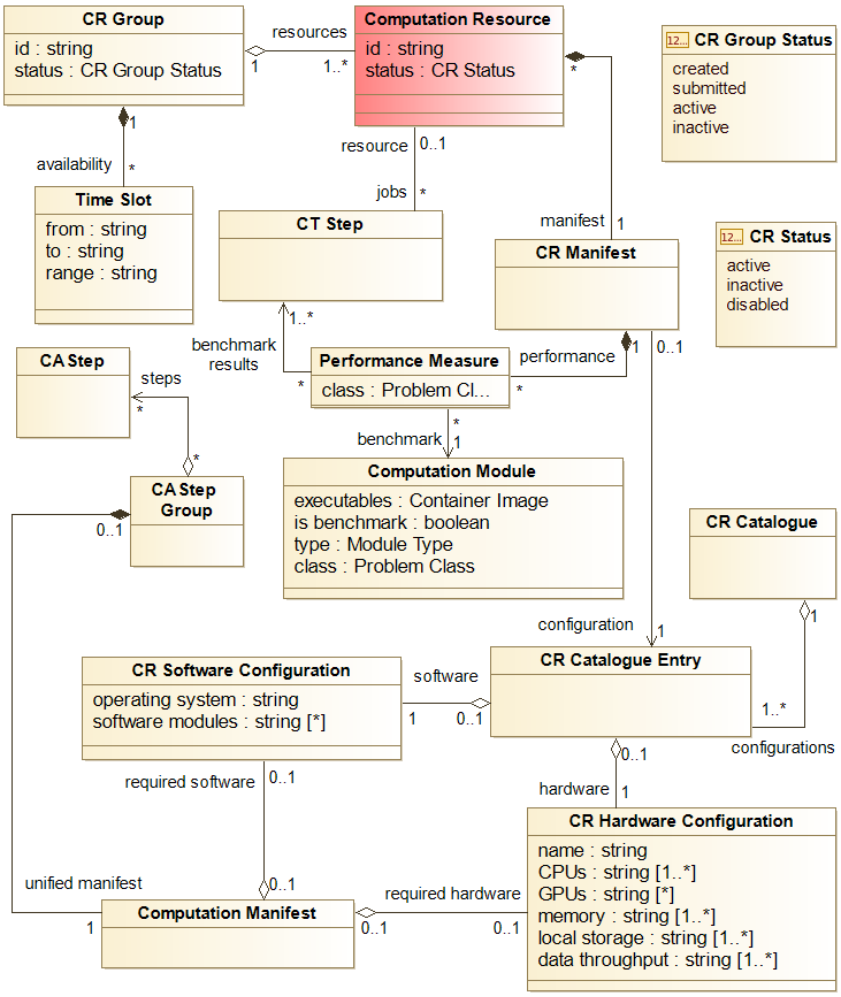

Fig. 2 Domain model for Distributed HPC resources

time to determine these values depends on the sequence of CA Steps and can also be determined by the user interactions (details are out of scope of this paper). The data collected by the computation runtime engine is needed to create future execution plans. In addition, the data stored in each CT can be used in the cases when the computation module has to be stopped in order to transfer it to another computing location (e.g. when the given Computation Resource has to be shut down).

As already indicated, Computation Tasks are executed through running their steps on a network of distributed Computation Resources. The appropriate model is presented in Figure 2. Each Computation Resource can run several jobs defined through CT Steps (compare also with Figure 1) Depending on the actual arrangement

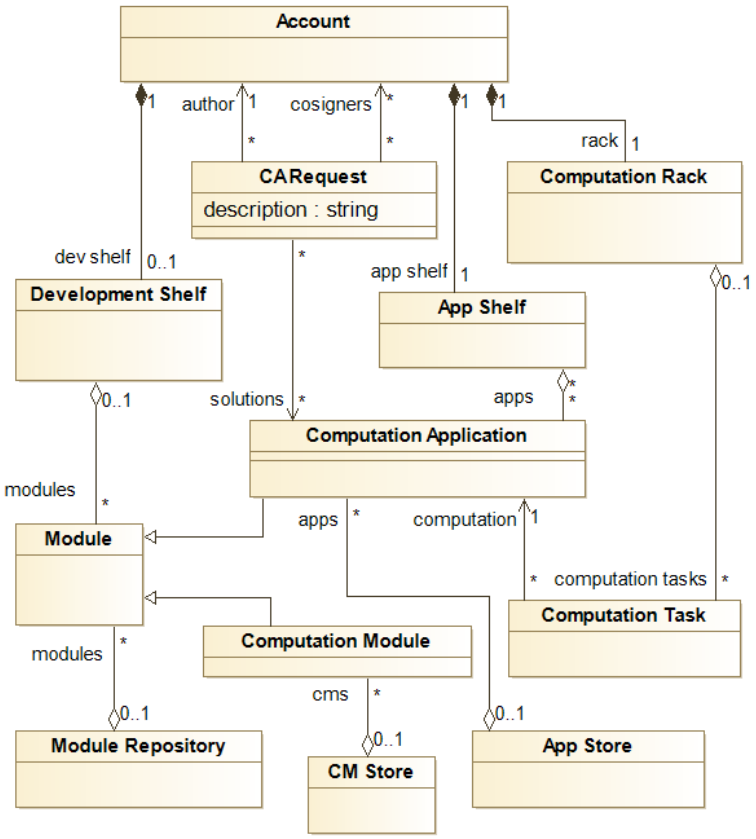

Fig. 3 Domain model for the Distributed HPC App Store

of data sources and parallelisation, jobs coming from a single Computation Task can be assigned to the same or different Computation Resources.

Similarly to Computation Modules, also CRs have their manifests (CR Manifest). The primary purpose is to collect and store information about the individual hardware configuration and available software within the computing node. These parameters can specify a single machine as well as a group of machines constituting a computational cluster. Within the manifest, virtual launch environments for containers (cf. Docker[21]) can also be defined. This solution significantly simplifies the management of the computing network. All configurations are assembled in a central catalogue (CR Catalogue) and contain descriptions of individual nodes of the calculation network. Each entry defines specific processor parameters or processors available for calculations, graphics processor parameters, available memory, disk resources, and essential operating system parameters within which the jobs assigned to a CR can be executed.

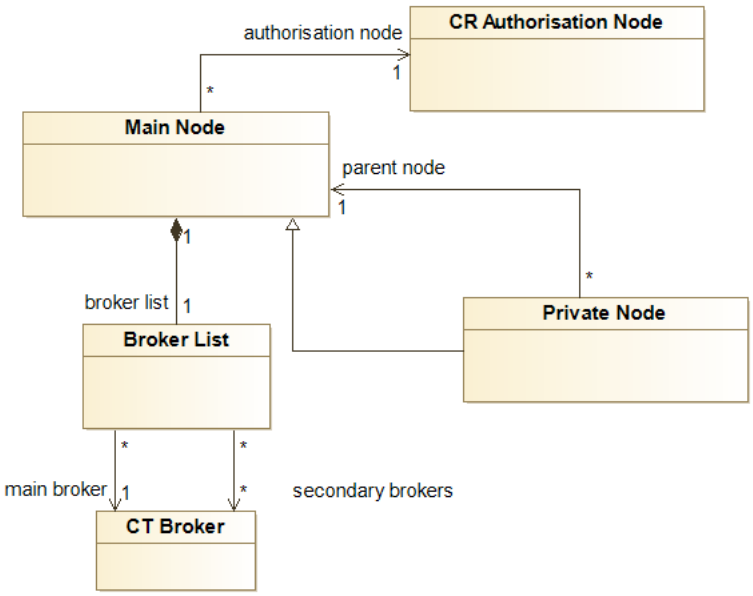

Fig. 4 Domain model for the Distributed HPC network

Apart from the manifest for a single CR, the CR owner can also define information on time slots during which computational tasks can be carried out. This can be defined for certain CR Groups. We can also notice that the manifest mechanism allows for collecting and managing information about the actual performance of computing nodes through benchmarking (see Performance Measure in Figure 2). Performance benchmarking is realized by executing specific calculation modules that serve as actual reference points for specific classes of computational problems (cf. Problem Class). Besides this mechanism, node performance can be calculated based on historical records of already performed calculation modules. What is important, the CR Manifest must be matched with the computational manifests attached to each CM. In this way, the computing network can determine if the CR meets the minimum requirements to run a specific CM. The information contained in manifests and the data on the declared time availability of nodes and information from performance tests are used by the mechanism that optimizes plans made for computational algorithms. Based on these plans, calculations can be assigned to the appropriate nodes.

The above computation-related elements have to be structured for storage and for presentation to the various actors of an DHCP system. Each element receives an appropriate metric containing basic data related to its purpose. This leads us to Figure 3 that presents such structuring elements. The main element that organises Computation Applications or its element is the App Store. It contains all the approved and available CAs. In order for an application to be used by an end-user or by an app-developer, it needs to be selected and added to the App Shelf, associated with as Account with proper credentials. All the Computation Tasks within an Account

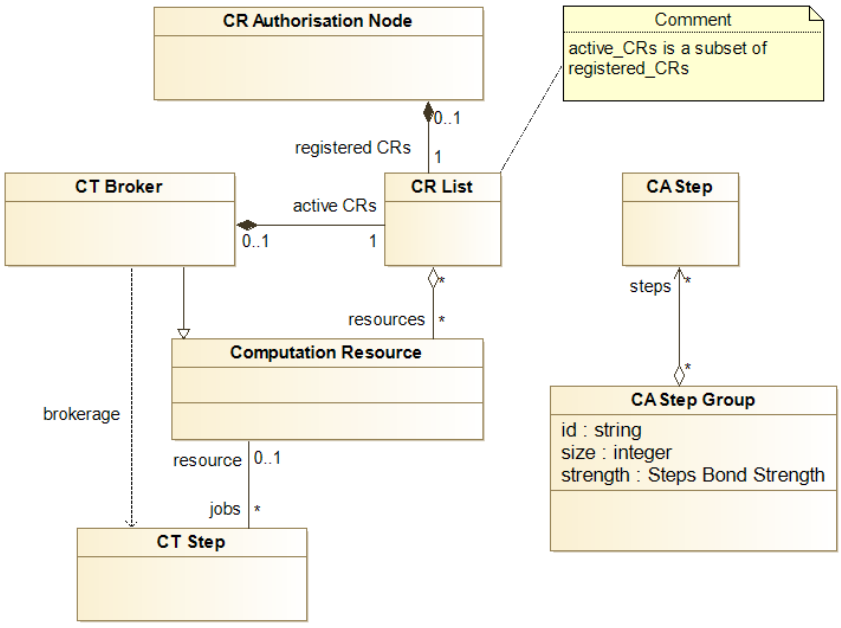

Fig. 5 Domain model for brokerage of HPC computations

are organised into a Computation Rack. From the presentation point of view, the two organising elements (App Shelf and Computation Rack) are shown in the form of a so-called Computation Cockpit.

In addition, for the end-user, who is the developer of computational applications, the Computation Cockpit provides access to the programming shelf. This shelf, in addition to ready-made calculation modules or their elements, contains the basic elements of the system in the form of shares or stocks. This mechanism is used to group all modules (CM and CA) developed in a given account. The application creator may also have unfinished modules over which development work is currently underway. All modules completed and sent by the developers are stored in the Module Repository. Each time a module is accepted, it is added to the App Store (CA) or the CM Store (CM). As part of the development account, apart from creating new modules, it is also possible to test them. A user with authorizations has the ability to run solutions on his own or test calculation nodes, bypassing module authorization. However, these activities are significantly reduced by the appropriate limits of the resources of the computing platform to limit fraud.

A complete Distributed HPC network is composed of potentially many Computation Resources. These resources have to be organised and efficiently brokered. For this reason, we introduce additional network elements, as shown in Figure 4. Main (Master) Nodes represent centrally managed machines which group sets of CRs and provide associated services to the end-users. The network may also contain Private Nodes which provide services only to restricted groups of users; for example for

development or testing purposes. Important elements in the network are CT Brokers. They are responsible for the process of "brokerage" – assigning CTs and their CT Steps to CRs during runtime. Additional details related to this mechanism are shown in Figure 5. It is worth mentioning that for the purpose of brokerage, CA Steps in an algorithm should be divided into CA Step Groups. These groups determine CT Steps that should be assigned to the same or closely located CRs.

4 Functional requirements model for distributed HPC

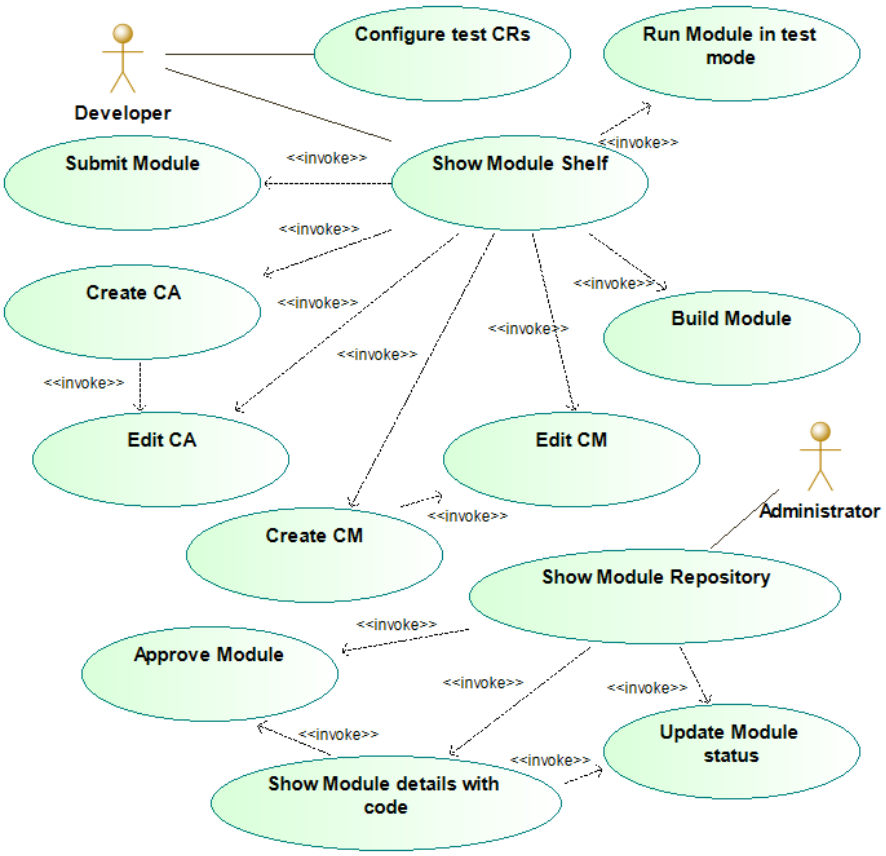

Fig. 6 Use case model for CA development

Based on the DHPC vocabulary defined through the domain model we can finally define functional requirements for Distributed HPC systems. This is illustrated through a use case model presented in Figures 6, 7 and 8. Computation Application development is centred around Module Shelves ("Show Module Shelf", see Figure

6). Independent developers can develop their CMs and CAs, and test them using test CRs ("Configure test CRs" and "Run Module in test mode"). To be used within the computation network, a module needs to be placed in the central Module Repository and approved ("Submit Module" and "Approve Module"). Developers can access and use these approved modules when developing their applications ("Edit CA"). As the pool of Computation Modules grows, building a Computation Application tends towards composing flows of ready blocks of computation algorithms.

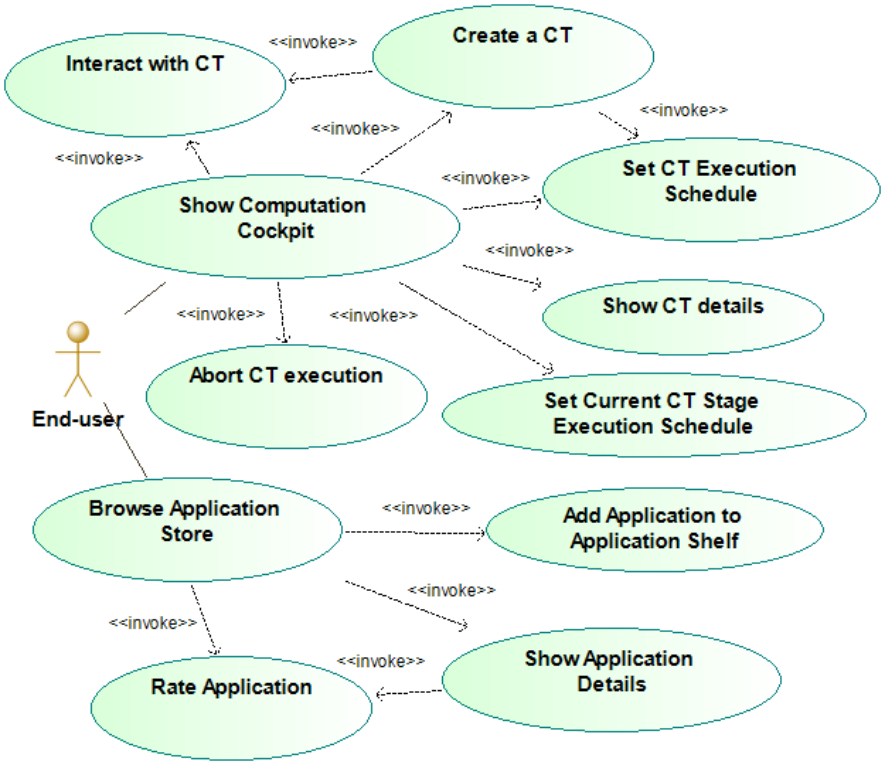

Fig. 7 Use case model for CA usage

All the centrally approved CAs are accessible to end-users through the App Store ("Browse Application Store", see Figure 7). The users can browse the store and place CAs in their local Application Shelfs ("Add Application to Application Shelf"). To run a CA, the user opens a Computation Cockpit and starts a Computation Task ("Create a CT"). It has to be noted that running a CA differs to running a typical interactive application. It is normally composed of a rather small amount of user interaction (entering parameters) followed by relatively long periods used to run specific CMs on assigned CRs. Thus, end-user interaction would become more asynchronous. Usually, the end-user would enter most of the computation parameters at the beginning ("Set CT Execution Schedule"). However, sometimes a

user interaction might be needed (as determined by the application flow) somewhere in the middle – between CM calls. In such cases, the Computation Cockpit needs to inform the user about the necessity to interact and allow to perform such interaction whenever the user is ready ("Interact with CT"). It can be noted that in such situations, the CAs would need to be put in a "sleep" mode in order to save computation resources.

To illustrate the functionality required from a DHCP system, as seen by the end-user, below we present a detailed scenario for the "Create a CT" use case. This use case is central from the point of view of further development of the underlying distributed computation mechanisms.

```
1. E-u selects Computation Application
2. E-u selects "Create Computation Task" button
3. System shows "CT Start" screen
4. E-u enters Computation Valuation
5. System invokes "Set CT Execution Schedule" use case
6. E-u selects "Confirm" button
7. System initiates Computation Task
8. [initial interaction with the user needed]
   System invokes "Interact with CT"
9. System executes Computation Task
```

As we can see, the end-user has to define the Computation Valuation data that determines priority of computations. It can be noted that the Computation Task already contains the manifest data that can be used by the DHPC system to distribute the computations to appropriate Computation Resources. If needed, the "Set CT Execution Schedule" use case allows to enter all the Data Source Addresses and Parameters in advance. Alternatively, appropriate interactions will need to be invoked ("Interact with CT") when required by the application code. After possible initial interaction, the system executes the Computation Task. Inside the system, this means that individual Computation Steps are automatically assigned to matching Computation Resources. This is done in accordance with the sequence and parallelisation of these steps within the algorithm of the particular CA.

Finally, the end- users also have the possibility to specify details of applications that are currently unavailable, but could be useful to them („Create CAR", see Figure 8). Such CARs (Computation Application Requests) are also browsable by other users („Show CARs List as End-User", „Show CAR Details as End-User"), which have the possibility to co-sign them („Cosign CAR"). Finally, when popular requests are implemented by the developers, they have the possibility to suggest their application as a solution to a specific CAR („Sugest CA for CAR").

Another set of requirements associated with management and supervision of CRs is shown in Figures 9 and 10. The end-users can access their CRs trough browsing owned CR Groups („Show owned CR Group list", see Figure 9). Then they have the possibility to execute certain actions an CR groups („Submit a CR Group", „Safely deactivate a CR Group") or go to managing a specific CR („Show own CR Group Resource list").

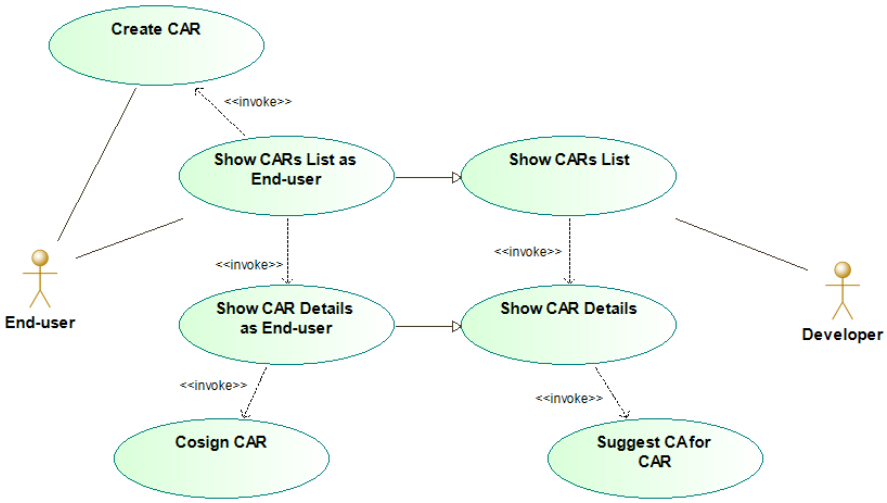

Fig. 8 Use case model for managing CA requests

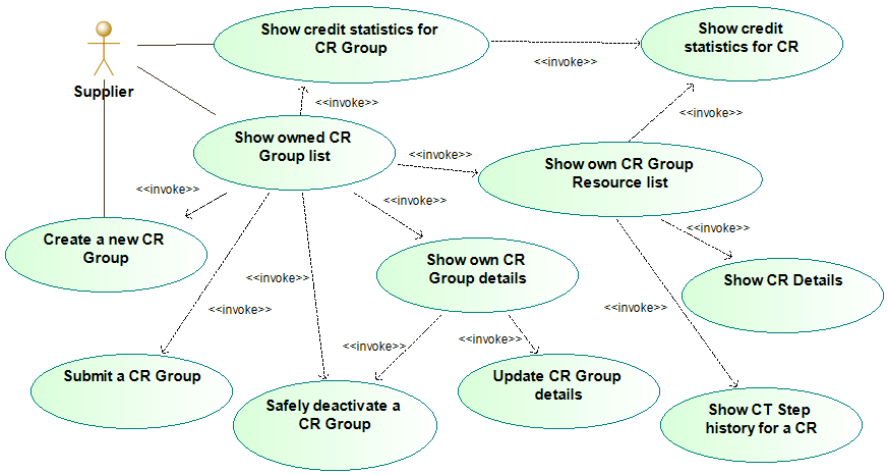

Fig. 9 Use case model for CR management

Supervision of CRs is based around a list containing all CR Groups („Show all CR Group list', see Figure 10'). Then, the Administrator can change resource statuses („Activate/deactivate a CR Group") or check the details of particular CRs („Show CR Group Resources list", „Show Advanced CR details") and run benchmarks on them („Execute Single Benchmark").

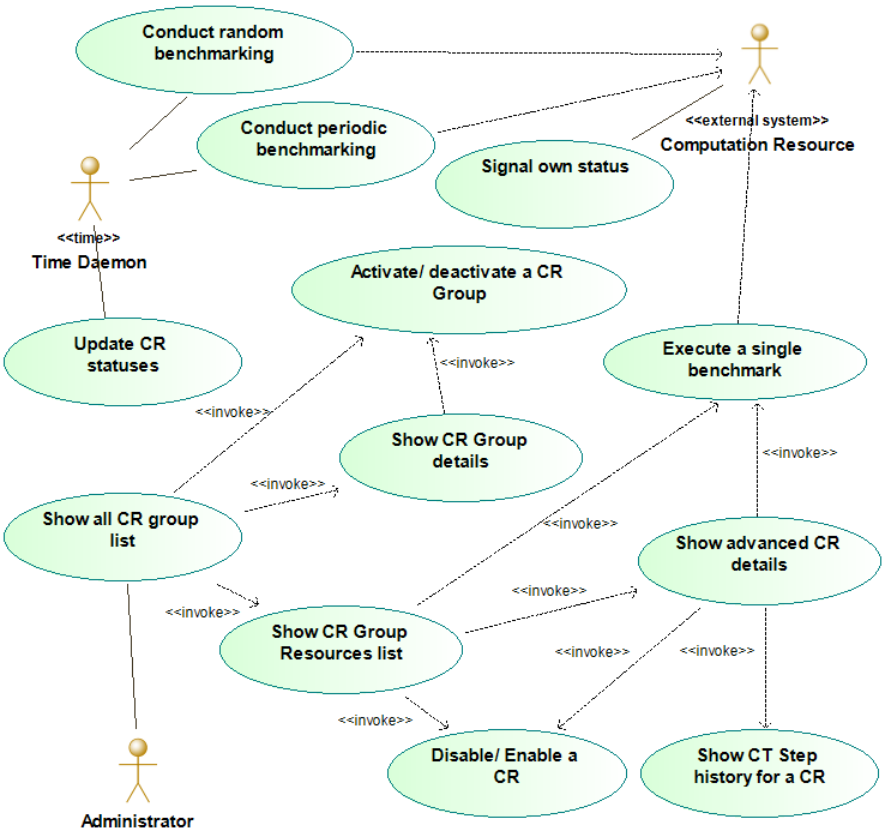

Fig. 10 Use case model for CR supervison

5 Reliability and security of calculations

As a supplement to the functional requirements, we can define key quality characteristics required from a Distributed HPC system: reliability and security. To assure high degree of reliability, practically every element of the computing platform needs to be duplicated or replaced by a spare element. For this purpose, additional units have to be established for all network elements. In addition, as part of the computing platform, a mechanism needs to be designed to allow automatic reconstruction of connections between network nodes in such a way as to maintain the consistency of the computing network.

The system, in its assumptions, needs to implement the transparency principle in the form of:

- **transparency of access** - unification of data access methods and hiding differences in their representation between computation modules,

- **position transparency** - users cannot determine the physical location of the resource except for the input data and the location of the calculation results,
- **transparency of migration** - the ability to transfer resources between servers without changing the ways of referring to them in the event of a computing node failure or termination of its availability,
- **transparency of movement** - resources can be transferred even during their use, primarily when calculations are performed in parallel,
- **transparency of reproduction** - hiding users from multiplying resources in the form of duplicated nodes,
- **transparency of concurrency** - the possibility of concurrent data processing does not cause inconsistencies,
- **failure transparency** - masking transient failures of individual components of the distributed system by transferring calculations to another calculation node,
- **transparency of durability** - masking the method of storing the resources in the computation network.

To achieve the desirable level of security, a unique mechanism for the authentication of computing modules needs to be designed. Each module created by the programmer must be approved by placing the applications in a batch of applications by the network administrators. The credibility of the calculation modules will be confirmed by the appropriately implemented hierarchical structure of the electronic signature. In addition to the security of the software, an appropriate procedure is proposed for adding new nodes to the computing network. The node must be manually added by network administrators, while its manifest must be prepared by the local node administrator (participant of the network) and verified by appropriate tests run on the newly added node. The node's data will also be confirmed by an appropriate electronic signature.

In addition to the security of calculation modules, it is necessary to ensure data security in the computing network. In order to protect data appropriately generated by computing modules, a blockchain-based blocking mechanism is envisioned. In this way, it will be possible to ensure reliability and integrity of data flow within the entire computing network. The blockchain mechanism will also be used to ensure settlements between participants of the computation network.

6 Summary and future work

Analysis of existing literature on DHPC domain models and functional requirements shows lack of direct approaches of this kind. Our main contribution is thus the presented requirements model, which is aimed at enabling the development of more coherent and universal DHPC systems in the future. We provide a unified vocabulary and functional characteristics, including a use case model with example scenario details that can be applied to a wide range of future DHPC systems. The presented models thus can serve as a unifying framework to join a heterogeneous set of Computation Resources into a coherent computation environment. Such a system

would involve a set of centrally managed machines (Master Nodes) that would allow to manage the development of Computation Applications, run Computation Tasks and manage Computation Resources. As future work, we also plan to develop a model for handling payments within the system. We should note that all the information required for this purpose is already available in the presented model. The challenge here will be to develop an appropriate scheme and technology that would take care of such distributed payments.

The core network of Master Nodes would be complemented by a growing set of Computation Resources, offered by various suppliers. Further research agenda would thus need to involve the development of a unified architecture (both physical and logical) for this new system. Challenges in this area involve designing a runtime engine for CAs that would involve brokerage of CMs as CT Steps.

The runtime engine would need to handle the execution of CA Steps, including CM calls and user interactions. This would need to be based on building a high-level language to define sequences of CA steps – we call it the Computation Application Language (CAL). We can observe that the presented domain model can be quite naturally transformed into a metamodel for CAL. This would allow to treat it as a domain-specific language and create a model-driven framework – a visual editor and code generator. The challenge here is to assure ease of use (even for non-IT personnel) and necessary levels of computation performance of the runtime engine for CAL. The Computational Application Language proposed by us can be adapted for any field, which allows the implementation of a domain-based, high-performance distributed computing platform.

Building a full computing environment based on the proposed solution would allow for performing highly-scale calculations in a fully distributed computational model using multiple computing nodes. Such a solution can be implemented both for large computing centres and for small teams that need short-term access to high computing power. The proposed solution may be in the future a base for creating a commercial computing network using distributed computing nodes. The designed solutions allow to scale the solution to various types of problems efficiently, and making them independent of the fixed domain allows for simple adaptation to any applications.

The above-introduced research agenda is currently underway as part of the BalticLSC project where the goal is to develop and validate the system consistent with the presented conceptual requirements models.

Acknowledgements This work is partially funded from the European Regional Development Fund, Interreg Baltic Sea Region programme, project BalticLSC #R075.

References

1. Armstrong, R., Gannon, D., et al.: Toward a common component architecture for high-performance scientific computing. In: Proceedings. The Eighth International Symposium

on High Performance Distributed Computing, pp. 115–124. IEEE Comput. Soc (1999)
2. Bernholdt, D.E., Allan, B.A., et al.: A component architecture for high-performance scientific computing. Int. J. of High Performance Computing Applications **20**(2), 163–202 (2006). DOI 10.1177/1094342006064488
3. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.D.: The role of trust management in distributed systems security. In: Secure Internet Programming, pp. 185–210. Springer Berlin Heidelberg (1999)
4. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. In: Proceedings 1996 IEEE Symposium on Security and Privacy, pp. 164–173. IEEE (1996)
5. Bryant, B.R., Gray, J., Mernik, M.: Domain-specific software engineering. In: Proceedings of the FSE/SDP workshop on Future of software engineering research, pp. 65–68. ACM (2010). DOI 10.1145/1882362.1882376
6. Bunch, C., Chohan, N., et al.: Neptune: a domain specific language for deploying HPC software on cloud platforms. In: Proceedings of the 2nd international workshop on Scientific cloud computing - ScienceCloud 11, pp. 59–68. ACM Press (2011). DOI 10.1145/1996109.1996120
7. Cappello, F., Djilali, S., Fedak, G., Herault, T., Magniette, F., Néri, V., Lodygensky, O.: Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. Future Generation Computer Systems **21**(3), 417–437 (2005). DOI 10.1016/j.future.2004.04.011
8. Dongarra, J., Sterling, T., Simon, H., Strohmaier, E.: High-performance computing: Clusters, constellations, MPPs, and future directions. Computing in Science and Engineering **7**(2), 51–59 (2005). DOI 10.1109/mcse.2005.34
9. Foster, I., Zhao, Y., Raicu, I., Lu, S.: Cloud computing and grid computing 360-degree compared. In: 2008 Grid Computing Environments Workshop, pp. 1–10. IEEE (2008). DOI 10.1109/gce.2008.4738445
10. Giles, M.B., Reguly, I.: Trends in high-performance computing for engineering calculations. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **372** (2014). DOI 10.1098/rsta.2013.0319
11. Grimshaw, A.S., Wulf, W.A., et al.: The Legion vision of a Worldwide Virtual Computer. Communications of the ACM **40**(1), 39–45 (1997). DOI 10.1145/242857.242867
12. Hager, G., Wellein, G.: Introduction to high performance computing for scientists and engineers. CRC Press (2010)
13. Hernández, F., Bangalore, P., Reilly, K.: Automating the development of scientific applications using domain-specific modeling. In: Proc. Second Int. Wrkshop on Software Eng. for High Performance Computing System Applications - SE-HPCS '05, pp. 50–54. ACM Press (2005). DOI 10.1145/1145319.1145334
14. Lampson, B., Rivest, R.: Cryptography and information security group research project: a simple distributed security infrastructure. Tech. rep., Technical report, MIT (1997)
15. Laure, E.: Distributed high performance computing with OpusJava. In: Parallel Computing: Fundamentals and Applications, pp. 590–597. PUBLISHED BY IMPERIAL COLLEGE PRESS AND DISTRIBUTED BY WORLD SCIENTIFIC PUBLISHING CO. (2000). DOI 10.1142/9781848160170_0070
16. Laure, E.: High level support for distributed high performance computing. Ph.D. thesis, University of Vienna (2001)
17. Li, Y.: DRUMS: Domain-specific requirements modelingfor scientists. Ph.D. thesis, Technische Universität München (2015)
18. Li, Y., Guzman, E., Bruegge, B.: Effective requirements engineering for CSE projects: A lightweight tool. In: 18th International Conference on Computational Science and Engineering, pp. 253–261. IEEE (2015). DOI 10.1109/cse.2015.49
19. Liu, H., Parashar, M.: Enabling self-management of component-based high-performance scientific applications. In: 14th IEEE International Symposium on High Performance Distributed Computing, HPDC-14, pp. 59–68. IEEE (2005)
20. Membarth, R., Hannig, F., et al.: Towards domain-specific computing for stencil codes in HPC. In: 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, pp. 1133–1138. IEEE (2012). DOI 10.1109/sc.companion.2012.136

21. Merkel, D.: Docker: Lightweight linux containers for consistent development and deployment. Linux Journal **2014**(239) (2014). URL http://dl.acm.org/citation.cfm?id=2600239.2600241
22. Moreira, J.E., Midkiff, S.P., et al.: Java programming for high-performance numerical computing. IBM Systems Journal **39**(1), 21–56 (2000). DOI 10.1147/sj.391.0021
23. Palyart, M., Lugato, D., Ober, I., Bruel, J.M.: MDE4HPC: An approach for using Model-Driven Engineering in High-Performance Computing. In: SDL 2011: Integrating System and Software Modeling, vol. 7083, pp. 247–261 (2011). DOI 10.1007/978-3-642-25264-8_19
24. Palyart, M., Ober, I., other: HPCML: A modeling language dedicated to high-performance scientific computing. In: Proc. 1st International Workshop on Model-Driven Engineering for High Performance and CLoud computing - MDHPCL12 (2012). DOI 10.1145/2446224.2446230
25. Ranjan, R., Benatallah, B., et al.: Cloud resource orchestration programming: Overview, issues, and directions. IEEE Internet Computing **19**(5), 46–56 (2015). DOI 10.1109/mic.2015.20
26. Schmidberger, M., Brugge, B.: Need of software engineering methods for high performance computing applications. In: 11th International Symposium on Parallel and Distributed Computing, pp. 40–46. IEEE (2012). DOI 10.1109/ispdc.2012.14
27. Teliba, H., Cisterninoa, M., Ruggierob, V., Bernardc, F.: RAPHI: Rarefied flow simulations on xeon phi architecture. Tech. rep., SHAPE Project (2016)
28. Van De Vanter, M.L., Post, D.E., Zosel, M.E.: HPC needs a tool strategy. In: Proceedings of the second international workshop on Software engineering for high performance computing system applications - SE-HPCS '05, pp. 55–59. ACM Press (2005). DOI 10.1145/1145319.1145335
29. Vecchiola, C., Pandey, S., Buyya, R.: High-performance cloud computing: A view of scientific applications. In: 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks, pp. 4–16. IEEE (2009). DOI 10.1109/i-span.2009.150