The Journal of Supercomputing (2021) 77:11866–11880
https://doi.org/10.1007/s11227-021-03731-6

Semantic tools for development of high-level interactive applications for supercomputers

Maxim Gorodnichev[1,2,3] · **Danil Lebedev**[4]

Accepted: 10 March 2021 / Published online: 30 March 2021

Abstract

The paper addresses the problem of devising a systematic approach and software tools to support development of interactive supercomputer applications on the basis of low level codes that are typically used on supercomputers for numerical simulation and data processing. An interactive application should help a user to systematically organize all the activities associated with solution of some class of problems on remote high performance computing systems. Activities include input data preparation, chaining of remotely run computing jobs, visualization, search and comparison of results, performance optimization and others. A platform for development of interactive supercomputer applications is proposed. The core of the platform is a visual language that allows a developer to formally describe activities (operations) and their relations to immutable data objects ("inputs" and "outputs"). Such a representation of a problem domain contains information about meaningful combinations of operations and becomes a basis for automated derivation of necessary user scenarios. A developer collects a library of UI components to represent data objects and a library of program modules that implement operations. These libraries are used in generation of a web-application that provides end users with appropriate interface to support derived scenarios.

Keywords Interactive supercomputer application · Usability · User interfaces · Domain specific language · Structural synthesis of programs

✉ Maxim Gorodnichev
maxim@ssd.sscc.ru

Danil Lebedev
danil.lebedev.0881@gmail.com

[1] Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russia

[2] Novosibirsk State University, Novosibirsk, Russia

[3] Novosibirsk State Technical University, Novosibirsk, Russia

[4] Astana IT University, Nur-Sultan, Kazakhstan

1 Introduction

Users of high performance computing (HPC) systems (supercomputers) either employ well established packages such as MATLAB, ANSYS, OpenFOAM, or develop their own codes in order to solve numerical simulation and/or data processing problems. The tools used by the latter group of users are rather low-level such as Fortran and C/C++ languages, mathematical libraries and tools used to implement parallelism (MPI, OpenMP, CUDA, MapReduce frameworks, etc) [1]. Usually, users of such codes organize computational experiments without any automation. That is, users have to manually upload codes and initial data to a target HPC system, build their programs, specify and submit computing jobs to some job management systems, monitor their completion. Should a user need a more complex experiment that requires a number of dependent jobs to be fulfilled in some order, the user has to repeatedly fulfill these actions manually for each step of the computational experiment or develop low-level scripts to run particular tasks in the required order. It means that all activities of a user and all data objects related to such computational experiments are out of control of any information system and a user bears full responsibility for running such experiments in a correct and efficient way. The problem becomes more severe if a number of different computing and storage systems have to be involved.

The objective of this work is to study how this problem can be addressed with a formal approach to management of computational knowledge described in [2]. The approach is based on the idea of structural program synthesis (Sect. 2). We create a software platform to assist development of interactive supercomputer applications. While the general concept of the structural program synthesis provides a sound basis for the platform design, an attempt to practical implementation of the concept in a specific application area reveals questions that were not covered in theoretical works.

A detailed example of a situation that requires development of an interactive supercomputer application can be found in [3], where data objects, operations, usage scenarios that are needed to study parameters of large-scale lattice gas automata are described and a manual implementation of an interactive supercomputer application is presented. The platform proposed in the present paper is used, particularly, to re-implement such an application in a more systematic, modular way that allows for easier future extensions, better maintainability, automated usage analysis and optimization.

2 Structural Program Synthesis

The method of structural program synthesis is described in [2] using the concept of a simple computational model. Formally, a simple computational model is defined as $C = (X, F, In, Out)$ where X is a finite set of variables, F is a finite set of functional symbols (operations), $In(a) = In_a \subseteq X$ is a set of m_a so called input variables defined for each $a \in F$, and $Out(a) = Out_a \subseteq X$ is a set of n_a so called output variables defined for each $a \in F$. This formal structure is interpreted in the domain D by associating a certain function $f_a : D^{m_a} \to D^{n_a}$ with each operation $a \in F$, and each variable $x \in X$ is interpreted with its value $I(x) = d_x \in D$. A correct interpretation is assumed, which means that output variables of each $a \in F$ are interpreted as if they were "computed" by function f_a given the values of the input variables of a.

Let $V \subseteq X$, $A \subseteq F$ be given. A set of functional terms $T(V, A)$ is defined as follows:

1. If $x \in V$, then x is a term $t \in T(V, A)$; the sets $in(t) = \{x\}$, $out(t) = \{x\}$ are called "input" and "output" variables of the term, respectively.
2. Let $a \in A$, $In(a) = \{x_1, ..., x_{m_a}\}$, be given and consider the term $t = a(t_1, ..., t_{m_a})$, where $\forall i \in 1..m_a : t_i \in T(V, A)$. The term $t \in T(V, A)$ if and only if $\forall i \in 1..m_a : x_i \in out(t_i)$. In this case: $in(t) = \cup_{i=1}^{m_a} in(t_i)$, $out(t) = Out(a)$.

We say that the term t "computes" the variable y if $y \in out(t)$. If we consider a subset $W \subseteq X$ and there is a set of terms $T(V, A)$ such that $\forall w \in W : \exists t \in T(V, A)) : w \in out(t)$ then $T(V, A)$ is an algorithm that allows one to compute the variables W being given the values of the variables V.

It is possible to set a computational problem on a model: with sets of variables V and W being given, it is required to deduce an algorithm, that computes the variables in W being given the values of the variables in V. Note, that more than one algorithm can be deduced (also, it is possible that no such algorithm can be deduced on the model).

If we have program modules (programs, procedures, network services, etc.) to implement functions associated with functional symbols, then it is possible to generate programs to implement such algorithms defined by sets of functional terms. The correct interpretation of the model C guaranties that implementation of any of such algorithms applied to the interpreted V will result in the same values of the variables from W.

A reasoner that deduces algorithms and a program generator can use additional information associated with functions, variables, program modules, and information about target computing systems to optimize selection of an algorithm and generation of a program according to non-functional requirements specified by a user.

We apply the idea of structural program synthesis to automate development of interactive supercomputer applications. We propose a software platform based on this approach that allows an application developer to formally describe the problem domain as a computational model. All the activities related to implementation of computational experiments are represented as operations of the model and data objects as variables. This information is used further to generate an interactive web-application to control computational experiments in a meaningful way.

3 Basic Ideas

The question is what are the tools needed to automate development of interactive applications capable of supporting required user scenarios? Generally speaking, a development environment should provide a developer with an appropriate language to allow one to formally describe user scenarios with all the details necessary to set a formal specification for automated generation of an interactive application. We consider a specific class of applications that help end users to organize relatively small number of relatively large computing operations into workflows, so this is not a general problem of arbitrary program generation. This allows us to expect that the number and diversity of the mentioned "details", the structure of data dependencies make the problem of application generation affordable. This point of view leads us to the following basic ideas on how to solve the addressed problem.

First, a unified language for management of computing jobs and data on different HPC resources is required. This is necessary to hide technical details from application developers (users as well) and to allow them to "speak" about job and data management at the appropriate abstraction level. We develop (Sect. 4) the HPC Community Cloud platform with its server providing a RESTful application programming interface (HPC2C API) to manage remote data and HPC jobs.

Then, in order to describe scenarios for computational experiments, an application developer needs a tool to describe unit operations such as "take these data, filter data (running an HPC job for that, the HPC2C API can be used here) and provide filtered data as a result" or "with these parameters of a reservoir, simulate a fluid flow within the reservoir". That allows a developer to create description of all data objects types and operations within a certain subject area. Having such elementary operations at hand, people usually think about composing scenarios from them. A scenario is a set of operations with a partial order defined on this set. One describes a number of scenarios and applies them whenever necessary. However, practice requires to break up such scenarios often. Sometimes we need to run some parts of such scenarios only or try certain steps of these scenarios many times while changing input parameters in order to obtain necessary behavior to move further. There are situations, in which data obtained within one scenario can be used in other scenarios. Such practice motivates us to think about a systematic support for such an activity. That can be achieved if we represent all meaningful compositions of elementary operations as a computational model (Sect. 2) that can be used to set problems on the model and derive scenarios to solve such problems. In order to support development and usage of computational models we develop an online integrated development environment (Sect. 5.1) that implements a visual language for representation of computational models.

With a computational model, a developer can set a certain problem on a model as described in Sect. 2. If we allow a developer to formally specify types of data objects that need to be supplied as inputs for a specific problem, then we can use this specification to generate input forms for the problem by composing basic input forms. A library of forms for basic types and rules for their composition becomes a part of the development environment. Complex input forms can be generated if a structure of data objects is known as a hierarchical collection of fields with specified basic types. For example, if a data object is a circle, it can be described with its radius *r* and coordinates of the center: *x*, and *y*, all these variables being of a real number type. This simple solution cannot be enough to build rich interfaces. The system should allow a developer to provide specific input solutions for certain data types. For example, if it is necessary to specify a geometry of a simulation domain (lets say, a reservoir), then a widget for visual construction of the domain can be included into the library of UI (user interface) components and associated with the "reservoire" type. The same ideas apply to visualization of data obtained as a result of some computational experiments.

As it was previously indicated, a fixed number of problem settings (user scenarios) can be not enough to respond to growing user demands. It means that, in some cases, an interface of a supercomputer application should also include the integrated development environment itself to allow a user to modify computational models and formulations of problems. Along with a set of problem settings selected for users by a developer, this will provide a flexible response to user requirements.

To conclude, an interactive supercomputer application, within a proposed approach, is a combination of numerical simulation and data processing codes that can be run remotely on HPC systems and a web-application that is generated on the basis of a computational model prepared by a developer (domain expert). Such an application offers an environment for users to run computational experiments in a comfortable way.

4 HPC community cloud

The objective of the HPC Community Cloud (HPC2C) project is to develop a platform able to accumulate knowledge about HPC-related content created by users and third-party developers: software development tools, data visualization tools, interactive training materials, network services, numerical simulation and data analysis software. The knowledge is accumulated in an active form: an interface should be provided for users to analyze the collected data, develop and run programs with the collected tools. The HPC2C project (Fig. 1) [4] implements a management server that provides external software systems with a RESTful application programming interface (API) and a web application that provides end users with a graphical interface. The HPC2C web interface is developed to improve the productivity of users of research and educational computer centers and, particularly, reduce the threshold for users to enter the field of HPC. A prototype of the HPC2C web application can be accessed at hpccloud.ssd.sscc.ru.

4.1 HPC2C management server

The management server keeps track of users, their files, registered remote computing resources, computing jobs. It organizes a storage for users' programs and data.

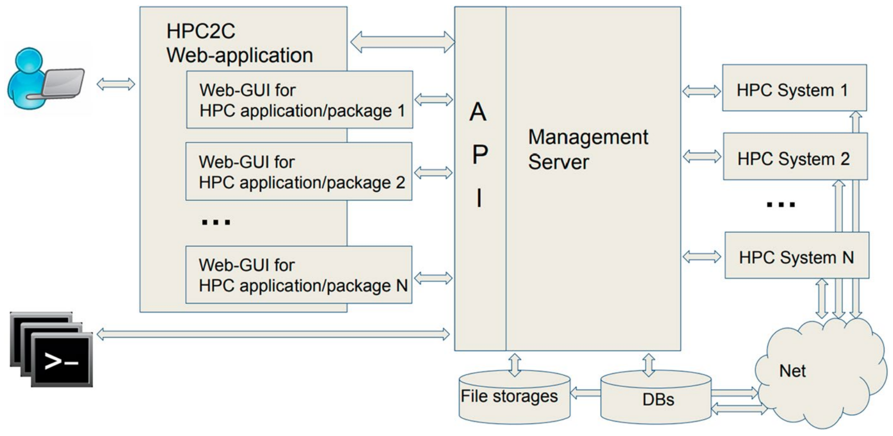


Fig. 1 HPC Community Cloud architecture. The HPC2C web-application "hosts" interactive supercomputer applications as plugins, they manage data objects and computing jobs on storage and HPC systems via the API provided by the management server. External systems/applications can use the API as well

HPC2C API is a unified basis for development of external software systems that can access the resources of computer centers for large-scale computations. The HPC2C API hides specifics of interfaces of particular computing systems behind a single access point and a single management system. This is achieved by implementing modules that transmit user commands made with a unified interface to specific interfaces of the controlled computing systems. Any user can register a computing system with the HPC2C by providing its address, access credentials (as in [5]), and a type of the interface. Examples of interface types are: a TORQUE job management system on a Linux cluster head node and a regular Linux box with no job management system.

Users can be included in different user groups. The rights to perform various operations on various objects can be set at the group level and at the level of individual users. Users can provide access to the objects they have created to other users and groups.

4.2 HPC2C user interface

The following possible usage scenario gives an idea of the HPC2C web-GUI interface. Also note, that all actions of the scenario can be commanded through the HPC2C API. The API is the basis for implementation of scenarios derived from computational models (Sect. 5.1). So, the following is a typical list of actions that are needed to run a computational experiment through the HPC2C interfaces. A user:

- uploads source files of a numerical simulation or data processing application to HPC Community Cloud,
- registers a computing system or selects a system from the list of systems provided by the HPC2C service and/or other users,

- describes rules for building programs based on the Make automation tool,
- builds programs according to the rules and corresponding to the architecture of the selected target computing system,
- uploads input data files,
- submits a computing job for execution,
- monitors the status of the jobs,
- gets access to the output files produced by the job and information on the errors (if there are any).

An application, once uploaded, can be registered with the HPC2C system and reused in further job submissions. In order to run such registered applications a user needs to provide input files and appropriate command line parameters.

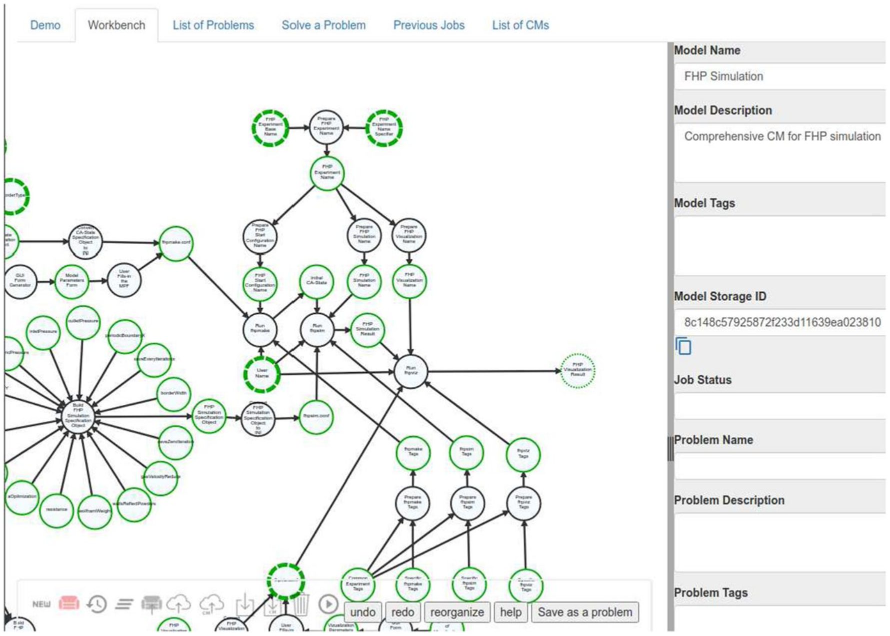
An application can be registered without uploading source files and building them through the HPC2C system. The only important point is that executable and configuration files are prepared according to the HPC2C rules for any computing system where users may want to employ the application. The invocation is done by the registered name of the application. The HPC2C server finds a version of the application that is appropriate for the target computing system and sends it for execution (this can be cached to avoid unnecessary transfers).

In other scenarios, users can submit jobs based on registered applications to which other users have granted access. All of these functions are available through the API, and end-users access these functions either through the HPC2C web interface or through external software systems.

5 Visual development environment

5.1 Overview

A development environment (Fig. 2) based on the proposed visual language (Sect. 5.2) is implemented as a part of the HPC2C web-application and is available online to the users of the HPC2C service. There are five main tabs: (1) the "workbench" tab is used to develop and represent computational models, specify computational problems on models, run computational jobs to solve specified problems and visualize running computations; (2) the "list of problems" tab contains a list of problems that have been specified on the model opened currently in the "workbench" tab; (3) selection of one of such problems opens the "solve a problem" tab where generated input forms for the selected problem can be found; (4) the "previous jobs" tab contains a list of computational jobs and selection of a job opens the "workbench" tab to visualize a computational model associated with this job and demonstrate either a final state of a finished job or visualize a live computational process by highlighting variables as they get their values computed; (5) the "list of CMs" contains a list of all available models. All the lists are searchable by names, tags or textual descriptions that can be associated with each of the objects (models,



Model Name
FHP Simulation
Model Description
Comprehensive CM for FHP simulation
Model Tags
Model Storage ID
8c148c57925872f233d11639ea023810
Job Status
Problem Name
Problem Description
Problem Tags

Fig. 2 Visual development environment. A workbench for visual construction of computational models

problems, jobs). Each stored object has a unique string identifier assigned by the system.

Users have access to the objects that they created or the objects shared by other users. An object can be shared by a user to the general public or to certain groups of users. Sharing of objects allows for collaboration.

The toolbox at the bottom contains controls to store, load models, to store problem specifications, run computational jobs, delete models, etc. A mouse together with keyboard key combinations are used to draw models.

5.2 Visual language

The workbench allows developers to draw computational models as they are defined in Sect. 2. Green circles (see Figs. 2 and 3) depict variables (also called "data objects"). Black circles depict operations. Variables and operations must have names, names are written in circles. A developer connects variables and operations with arrows to define input and output variables for operations. This is how a user can define a simple computational model. However, simple models are not enough to express many algorithms. Other approaches such as [6] introduce more powerful language constructs explicitly. Currently, we avoid this by hiding predicates and recursive calls in the implementation of the operations. Predicates work in the following way: an implementation of an operation depending on the input may set values of output variables or leave (some of them or all) not computed. The execution

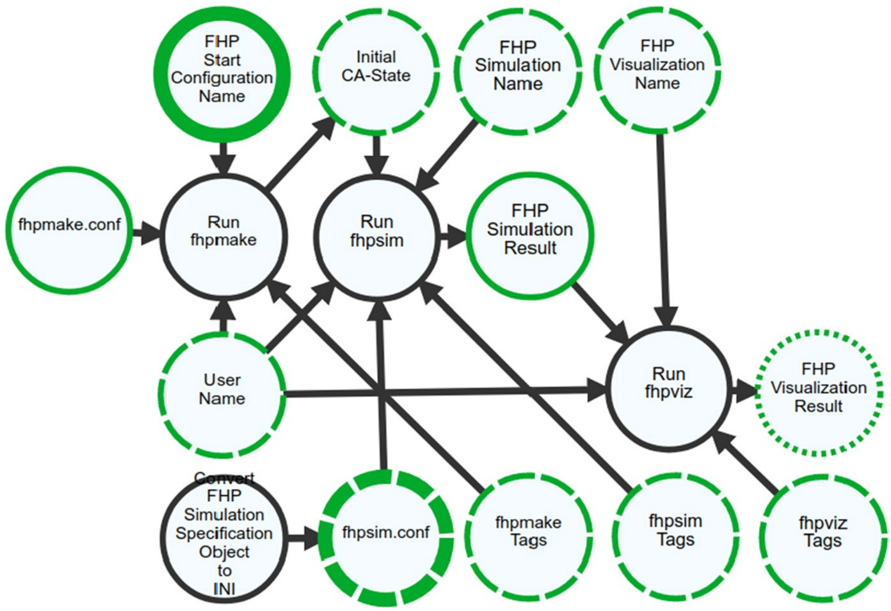


Fig. 3 Representation of a simple computational model in a visual development environment: variables are represented as green circles, operations are represented as black circles. The dashed outline (e.g., variables "fhpsim.conf" and "Initial CA-State") means that a variable belongs to the set V (see Sect. 2). The dotted outline ("FHP Visualization Result") means that the variable belongs to the set W. The thick outline (e.g., "fhpsim.conf") means that the variable was assigned a value. This is a part of a model developed to describe simulations with lattice gas automata [3]

system can notice that an operation finished without producing outputs. Such a solution does not allow a deduction system (a reasoner) to see predicates and use this information for workflow derivation. Recursive calls can be made visible for static analysis (see description of operation types below) but can also be hidden in a low-level procedure that implements an operation.

The following actions require that a developer opens a dialogue window associated with that or another object (variable or operation) and edits the properties of the object.

A developer has to associate an implementation procedure with each operation so that an execution system is able to fulfill operations automatically. The platform supports an extensible set of operation types. Currently, the following types can be chosen:

- JavaScript (JS) operations: a developer provides a piece of a JavaScipt code that implements the operation; a mapping between JS variables and model variables must be defined;
- HTTP methods: a developer may set the operation type as HTTP GET, for example, and provide a target URL and request parameters; this allows to directly access web-services where no complex preparation of a request is required (complex requests can be prepared within JS operations);

- “structured” operation: a computational problem is specified (Sect. 2) on some stored model; implementation of such an operation means that, first, an algorithm must be derived on the model and, second, the algorithm must be executed by the execution system; the same model can be used for problem specification—this allows to implement recursion that will be “visible” to the static analysis.

In fact, the reasoner and the execution systems are implemented as web services, so it is possible to call them to solve problems specified on the models from operations of the first two types. Such invocations will not be available for static analysis (unless explicitly specified in the description of non-functional properties of these operations).

A developer can set optional attributes of the variables:

- value: a number, an arbitrary string or a JSON-object can be set as a value of a variable; variable with a value is highlighted with a thick outline;
- JSON-Schema: is used to automatically check values and generate input forms;
- input: can be set to true or false;
- output: can be set to true or false.

The attributes “input” and “output” are used to specify a problem on a model. A developer sets such attributes for some of the variables. All the variables marked with $input = true$ constitute the V set (Sect. 2) (visualized with dashed outline) and all the variables marked with $output = true$ constitute the W set (dotted outline). When these sets are specified, the problem is specified and can be stored for further reference (becomes accessible on the “list of problems” tab). There are two options to run a specified problem: by setting values of input variables using dialogue windows on the “workbench” tab, or switch to the “solve a problem” tab and use generated input forms to set the values.

A developer can implement a GUI widget that can be used in a user interface to visualize the value of a certain variable and a GUI widget that can be used in a user interface to allow a user to set the value of the variable. Such widgets must be added to the library of widgets and associated with the computational model and the variable. There can be more than one widget of each type for each variable.

The execution system can check the value against a specified JSON-Schema automatically, but if a user needs more complex check then a developer (can be the same person) may wish to add a procedure into the library of “validators” that will be called automatically to check the value of a variable.

Operations and variables can also have any additional attributes (metadata) that a developer associate with these objects. Such information can be used by procedures that implement operations.

6 Storage

Models are stored as JSON-objects. There is a document-oriented database controlled by the platform. It is used by default to store models developed under the visual development environment. Also, it stores results of computations. Note, if some problem is set on a model and some computation is fulfilled, then some values of some variables become known (computed). As a result, a model with some of its variables being attributed with their values is stored (separately from the original model). Also, some other attributes can be set by operations and/or the deduction and execution systems, for example, to save performance profiling information.

One may note that a collection of such attributed models can be viewed as a table with variable values, or more generally as an OLAP cube (taking into account that variables and operations can have multiple attributes). In such a way, well-established tools can be used to analyze these collections of data. This provides another possibility for an end user to manage large volumes of data associates with computations.

Other types of storage can be used to store models, problem specification and other objects. Remotely stored objects can be references whenever needed with URIs. For example, a model can be fetched from a Git repository during execution of a workflow.

7 Real-life complications

Practice introduces complications that are not explicitly taken into account by the formalism described in [2] (Sect. 2). For example, it assumes that interpretations are correct in a sense that for any derivable algorithm that computes a certain variable from a given set of input variables the computed value of the output variable will be the same (Sect. 2). In real settings, specific definitions of equality can be important. In some cases we can tolerate “slightly” different values, e.g., if the values of a real number variable computed by two algorithms are different due to round-off errors. In other case, there can be a wrong procedure associated with some operation and the algorithm involving such an operation will produce an incorrect value at least on some inputs. Generally speaking, an expert is required to check adequacy of the results. An expert can specify rules that allow the system to consider two values as belonging to some equivalence class, specify a tolerable level of round-off errors, etc. On the other hand, one should note, that the algorithms derived on a computational model and their results can be compared automatically to some extent.

Another example is the case of an unsolvable problem specification. In practice, if a problem specification is given that does not allow the reasoner to derive an algorithm, it is not enough to provide a user with a negative result. The proposed platform advises the user to introduce some more input variables into the problem specification. Development of algorithms to provide an appropriate system reaction deserves a separate research. Similar situation is addressed, for example, in [7].

Definitions of a computational model and its interpretation assume that, for each operation, values of its output variables depend on values of the input variables only. Nothing is said about side effects of the procedures used to implement the operations. With HPC applications, when multiple independent data sources, storage systems and computing resources are involved, it is not possible to put all data and computing resources under exclusive control of the proposed platform. That means that variables managed by the platform will often serve as references to actual data object stored elsewhere. So, side effects and external factors can influence the results of computations.

As was mentioned in the Sect. 6, the history of computations associated with some computational model can be stored for future analysis. For non-trivial problem domains, the practice may require developers/users to reconsider previously developed and used models. Models can be restructured, extended, split, joined to other models; variables and operations can be renamed, etc. We need to be able to compare problem solutions obtained with different versions of a model. The platform should be able to assist a developer/user to keep track of the model changes and resolve such issues.

It is important to notice that, for non-trivial domains, an evolving, adapting computational model is not just a tool for development of an interactive supercomputer application but becomes a part of the application, the part that must be accessible to experienced users.

8 Related work

The project described in the paper relates to different fields of research.

The basic idea of the HPC Community Cloud (provide an API for management of remote HPC jobs) is shared by many other project devoted to construction of web-portals for HPC. An extensive survey of web-portals for HPC has been done recently in [8]. The problem of creation of such portals has a long history of discussion and a vast list of associated projects. During the last decade a number of projects emerged [4, 5, 9–12] building such portals around services that provide application programming interfaces (API) in the trending RESTful architectural style. The APIs allow developers to implement applications capable of data management and control of computing jobs on remote HPC resources. Having their specific traits, these projects seem to share a view on users' needs and associated problems, and, thus, are similar in approaches and solutions. The particular focus of the HPC Computing Cloud project [4] used and developed in the present work is on creation of a platform for accumulation and reuse of user-developed applications and collaboration of users over the development and use of the applications.

Besides the need for high-level interfaces for particular remote running applications, research in the field of unified interfacing systems for development, deployment and use of simulation and data processing applications is motivated by many factors. These factors include concerns related to reproducibility, accessibility, and transparency of computational research [13–15], problems of application discovery [16], need for collaborative work, diversity of target high-performance computing systems that must be abstracted for a regular application user, and others.

The development of the high-level HPC services should be based, among the other, on analysis of use cases as particularly underlined in [17]. This is actually the basis of our project because we develop a tool to specify use cases and generate high-level services according to such specifications.

The chosen approach to generation of efficient workflows and their execution is known as structural program synthesis. Other works in this field are devoted to the logical grounds of knowledge representation and derivation of algorithms or application of this concept to other problem areas [6, 7]. Our project is focused on practical problems and technical issues that are usually abstracted away in theoretical papers and particularly addresses the problem area of HPC applications. One of such practical problems is that non-functional requirements should be taken in account when deducing an algorithm (a workflow) on a computational model (so, this is not enough to just deduce some algorithm that matches functional requirements). This problem was formulated and addressed for example in [18]. A survey of approaches to program synthesis can be found in [19].

Composition and execution of workflows are studied extensively. There is a lot of research [20] devoted to scheduling of HPC workflows and there is a number of well-established (ProActive Workflows & Scheduling) and emerging systems that provide users with interfaces that allow to represent such scenarios in a form that can be interpreted by a system. Explicit representation of scenarios allows such systems to have control over workflows, use this information for better resource scheduling, resolve failures in a meaningful way and so on. The difference with our project is that we derive workflows. We do not develop new scheduling algorithms within the project, but rather follow the basic project idea to collect and apply existing algorithms.

9 Conclusion

A platform for development of interactive supercomputer applications is proposed. An interactive supercomputer application is a web-application that provides a high-level interface for a user and runs numerical simulation and data processing codes remotely on HPC systems to solve the problems that the user specifies through a high-level interface. The platform is built around a visual language that allows a developer to formally describe operations (user actions or computations) and their relations to immutable data objects (that either provide input information for the operations or are created as a result of the operations). Such a representation (a computational model) implicitly contains information about all meaningful user scenarios. Actually, it defines all necessary terms and their relations to allow a developer to "speak" at the level of the subject domain, thus defining a domain specific language. It provides a basis for automated derivation and implementation of necessary user scenarios and generation of a web-application that provides end users with appropriate interface to support derived scenarios. An extensible library of UI components is used in generation of interfaces. An extensible library of program modules that are needed to implement operations is used to automatically fulfill derived scenarios. Thus, the platform allows its users to accumulate and reuse knowledge about HPC application components in a systematic way.

Future work includes development of system algorithms capable of taking into account non-functional requirements: algorithms for workflow deduction, workflow scheduling and others that will help, particularly, to resolve issues mentioned in Sect. 7. Interestingly, such algorithms can be accumulated and used by the platform on the same principles as user-level application components.

Acknowledgements This research has been funded by the Science Committee of the Ministry of Education and Science of the Republic of Kazakhstan BR05236340 “Creation of high-performance intelligent analysis and decision making technologies for the “logistics-agglomeration” system within formation of the Republic of Kazakhstan digital economic” and under state contract with ICMMG SB RAS (0251-2021-0005).

References

1. Amaral V, Norberto B, Goulão M, Aldinucci M, Benkner S, Bracciali A, Carreira P, Celms E, Correia L, Grelck C, Karatza H, Kessler Ch, Kilpatrick P, Martiniano H, Mavridis I, Pllana S, Respício A, Simão J, Veiga L, Visa A (2020) Programming languages for data-Intensive HPC applications: a systematic mapping study. Parallel Comput. https://doi.org/10.1016/j.parco.2019.102584
2. Malyshkin V (2015) Active knowledge, LuNA and literacy for oncoming centuries. In: Bodei C, Ferrari G, Priami C (eds) Programming languages with applications to biology and security. Lecture notes in computer science, vol 9465. Springer, Cham. https://doi.org/10.1007/978-3-319-25527-9_19
3. Gorodnichev M, Medvedev Yu (2019) A web-based platform for interactive parameter study of large-scale lattice gas automata. In: Malyshkin V (eds) Parallel computing technologies. PaCT 2019. Lecture notes in computer science, vol 11657, pp 321–333. Springer, Cham. https://doi.org/10.1007/978-3-030-25636-4_25
4. Gorodnichev M, Vaycel S (2014) Organization of access to supercomputing resources in the HPC community cloud. Bulletin of the South Ural State University Series. Comput Math Soft Eng 3(4):85–95. https://doi.org/10.14529/cmse140406
5. O’Leary P, Christon M, Jourdain S, Harris C, Berndt M, Bauer A (2015) HPCCloud: a cloud/web-based simulation environment. In: IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), Vancouver, BC, pp 25–33. https://doi.org/10.1109/CloudCom.2015.33
6. Novikov FA, Novoseltsev VB (2010) Interpretable program specification language. Program Comput Softw 36:48–57. https://doi.org/10.1134/S036176881001007X
7. Giedrimas V, Omanovic S, Grigorenko P (2017) The evolution of automated component-based software development tools: from structural synthesis of programs to behavioral types. In: 2017 International conference on information science and communications technologies (ICISCT). https://doi.org/10.1109/icisct.2017.8188570
8. Calegari P, Levrier M, Balczyński P (2019) Web portals for high-performance computing: a survey. ACM Trans Web 13(1):5:1–5:36. https://doi.org/10.1145/3197385
9. Cholia S, Sun T (2015) The NEWT platform: an extensible plugin framework for creating ReSTful HPC APIs. Concurr Comput Pract Exper 27:4304–4317. https://doi.org/10.1002/cpe.3517
10. Bychkov IV, Oparin GA, Bogdanova VG, Pashinin AA, Gorsky SA (2017) Automation development framework of scalable scientific web applications based on subject domain knowledge. In: Malyshkin V (eds) PaCT. LNCS vol 10421. Springer, Cham. https://doi.org/10.1007/978-3-319-62932-2_27
11. Afanasiev A, Sukhoroslov O, Voloshinov V (2013) MathCloud: publication and reuse of scientific applications as restful web services. In: Malyshkin V (ed) PaCT 2013. LNCS, vol 7979. Springer, Berlin, Heidelberg, pp 394–408. https://doi.org/10.1007/978-3-642-39958-936
12. Sukhoroslov O, Volkov S, Afanasiev A (2015) A web-based platform for publication and distributed execution of computing applications. In: 14th International symposium on parallel and distributed computing, Limassol, pp 175–184. https://doi.org/10.1109/ISPDC.2015.27
13. Goecks J, Nekrutenko A, Taylor J (2010) Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol 11(8):R86. https://doi.org/10.1186/gb-2010-11-8-r86
14. Stodden V, Seiler J, Ma Z (2018) An empirical analysis of journal policy effectiveness for computational reproducibility. In: Proceedings of the national academy of sciences of the United States of America, vol 115(11), pp 2584–2589. https://doi.org/10.1073/pnas.1708290115

15. Jiménez RC, Kuzak M, Alhamdoosh M et al (2017) Four simple recommendations to encourage best practices in research software. F1000Research 6:876. https://doi.org/10.12688/f1000research.11407.1
16. Hucka M, Graham MJ (2018) Software search is not a science, even among scientists: a survey of how scientists and engineers find software. J Syst Softw 141:171–191. https://doi.org/10.1016/j.jss.2018.03.047
17. Struckmann N et al (2018) MIKELANGELO: MIcro KErneL virtualizAtioN for hiGh pErfOrmance cLOud and HPC Systems. In: Mann Z, Stolz V (eds) CCIS vol 824. Springer, Cham, pp 175–180. https://doi.org/10.1007/978-3-319-79090-915
18. Valkovskii VA, Malyshkin VE (1988) Synthesis of parallel programs and systems on the basis of computational models. Nauka, Novosibirsk (In Russian. Sintez parallel'nykh programm i sistem na vychislitel'nykh modelyakh)
19. Bodik R, Jobstmann B (2013) Algorithmic program synthesis: introduction. Int J Softw Tools Technol Transf 15:397–411. https://doi.org/10.1007/s10009-013-0287-9
20. Wu F, Wu Q, Tan Y (2015) Workflow scheduling in cloud: a survey. J Supercomput 71(9):3373–3418. https://doi.org/10.1007/s11227-015-1438-4