Available online at www.sciencedirect.com

ScienceDirect

Computer Physics Communications 178 (2008) 186–198

Computer Physics Communications

www.elsevier.com/locate/cpc

Metadata management for distributed first principles calculations in *VLab*—A collaborative cyberinfrastructure for materials computation

Pedro R.C. da Silveira [a,*], Cesar R.S. da Silva [a], Renata M. Wentzcovitch [a,b]

[a] *Minnesota Supercomputing Institute, University of Minnesota, Minneapolis, MN 55455, USA*
[b] *Department of Chem. Eng. & Mat. Sci., University of Minnesota, Minneapolis, MN 55455, USA*

Received 15 May 2007; received in revised form 27 August 2007; accepted 4 September 2007

Available online 21 September 2007

Abstract

This paper describes the metadata and metadata management algorithms necessary to handle the concurrent execution of multiple tasks from a single workflow, in a collaborative service oriented architecture environment. Metadata requirements are imposed by the distributed workflow that calculates thermoelastic properties of materials at high pressures and temperatures. The scientific relevance of this workflow is also discussed. We explain the basic metaphor, the receipt, underlying the metadata management. We show the actual java representation of the receipt, and explain how it is converted to XML in order to be transferred between servers, and stored in a database. We also discuss how the collaborative aspect of user activity on running workflows could potentially lead to race conditions, how this affects requirements on metadata, and how these race conditions are precluded. Finally we describe an additional metadata structure, complementary to the receipts, that contains general information about the workflow.
Published by Elsevier B.V.

PACS: 91.40.Ac; 91.30.Ab; 61.43.Bn

Keywords: First-principles calculations; Computational materials science; Distributed computing; Grids; Grid computing; Metadata; Metadata management

1. Introduction

Computational studies requiring sampling of points in parameter space appear in several scientific fields such as environmental sciences, astrophysics, particle physics, computational fluid dynamics, and materials science, the subject of our special interest. Usually these studies comprise a large number of tasks, each one responsible for a given parameter domain. As a common characteristic, the number of parameter points to be sampled grows exponentially with the number of parameters, and the workload usually grows accordingly. Fortunately, calculations related to different parameter domains are decoupled or, at least, loosely coupled. This enables them to be executed concurrently. Moreover, coarse grained scientific applications can be efficiently executed on Grids, such as the TeraGrid, or other distributed resources provided they are not strongly coupled.

Calculating elastic constants of Earth forming materials at high pressures and temperatures, $C_{ij}(T, P)$, from first principles is an important application in materials science. It is also one of the main scientific drivers of the ***VLab*** project [1]. The calculation is structured as a multi-phase workflow, each phase requiring extensive parameter sampling. Parameter point samples are pressure $\{P_i\}$ requiring $\sim 10^1$ points, strains $\{\epsilon_j\}$ ($\sim 10^{1-2}$), and phonon wave vectors in the Brillouin zone $\{\mathbf{q_n}\}$ ($\sim 10^1$), the q-points. Therefore, the cardinality of $\{P_i\} \times \{\epsilon_j\} \times \{q_n\}$ can be as high as $\sim 10^4$. Each point (P_i, ϵ_j, q_n) spans a single compute-intensive task, i.e., a full fledged DFT based ab initio calculation, requiring as much as $\sim 10^{16}$ floating point operations. Fortunately, all calculations in each phase can be decoupled and executed concurrently. Moreover, each phase generates substantial number of output files scattered throughout the distributed computing resources. These files need to

[*] Corresponding author.
E-mail address: pedros@msi.umn.edu (P.R.C. da Silveira).

0010-4655/$ – see front matter Published by Elsevier B.V.
doi:10.1016/j.cpc.2007.09.001

be recollected between phases in order to generate the inputs for the next one. An overview of the methodology to compute C_{ij} will be given in Section 2. Distributed computing, particularly Grid computing, is enjoying a growing acceptance in computational physics [2,3] and geophysics [4,5]. The ***VLab*** project aims to develop a novel service oriented architecture (SOA) [6] cyberinfrastructure (SOAC) designed to be a facilitator in performing this kind of calculations, among others. The SOAC is required to manage tasks in a distributed environment, leveraging the high throughput of distributed resources, in addition to the high performance of its constituent hosts. Additionally, it must provide tools, which are interfaced to the user through a portal, to manage execution, monitoring, and steering of workflows, data analysis, and visualization in a collaborative way.

Although the main motivation for ***VLab*** is the calculation of $C_{ij}(T, P)$, the resulting infrastructure is of general use, since it must support algorithms that are prototypical of several other types of parameter sampling calculations. Moreover, the calculation of a material's C_{ij} encompasses the calculation of its equation of state (EOS), a result of very broad interest: the knowledge C_{ij}s and EOS [7,8] of Earth forming materials are essential to advance our understanding of the physics of Earth's deep interior. The main source of geophysical data on those regions is seismic tomography. This technique offers 3D maps of shear and longitudinal seismic wave velocities throughout the whole Earth [9]. These velocities depend on the density and elastic properties of the medium they traverse. Available density maps [10] or 1D profiles [11] permit the conversion of velocity maps into maps of elastic moduli (shear and bulk moduli). This information can be translated into local chemical/mineralogical/thermal [12] models if elasticity data for Earth's forming minerals is available. Therefore, geophysical and planetary interiors' studies are directly or indirectly related to the elastic properties (equations of state (EOS), $P(V, T)$, and elastic constants $C_{ij}(P, T)$; with $P =$ pressure and $T =$ temperature) of materials. When these are not available, it is necessary to calculate them. One of the main goals of ***VLab*** project is to develop a facilitator for these kinds of calculations. This paper is primarily concerned with the description of the metadata and metadata management algorithms necessary to manage distributed execution of the tasks participating in multi-phase workflows involving data recollection between phases. Metadata are the bookkeeping information about data. Here data refers to information representative of an actual object, e.g. physical properties of a system. Data is usually structured. If the structure can change dynamically, it becomes convenient to keep data unstructured, or loosely structured, and represent the logical structure as metadata. An example is the linked cell structure common in molecular dynamics (MD) and Monte Carlo (MC) calculations. Atomic coordinates could be represented as five-fold indexed arrays, three for the cell, one for the atoms inside each cell, and one for the coordinate. Instead, linked lists whose elements point to actual data (metadata) are used to represent the structure, which is more economical and faster.

To take advantage of concurrency in parameter sampling applications, the parameter domain must be decomposed and mapped to several tasks. In traditional tightly coupled parallel systems, parameter domains are directly mapped to nodes and metadata is usually simple. In distributed grids, however, specific computing resources are abstracted, direct correspondence between domains and resources is lost, and carefully designed metadata becomes essential. Moreover, ***VLab*** must allow for collaborative interactions of multiple users with running applications, which imposes additional requirements on metadata.

In Section 2 we describe the algorithms used to decouple some calculations in the EOS generation workflow. As the requirements of this algorithm underlie several decisions regarding system structure and metadata management, a preliminary discussion on this point is provided. We then describe the complete C_{ij} calculation workflow in Section 2.1. In Section 3.1 we define the terminology and describe the basic metaphor underlying the metadata management. Section 4 gives a block description of the system. Section 5 discusses the main aspect of the metadata implementation and handling. A closing summary is given in Section 6.

2. Methodology

A particularly accurate method [13] to calculate EOS and C_{ij}s involves the use of damped variable cell shape molecular dynamics (VCS-MD) [14–17] with forces and stresses calculated from first principles [18–21]. In order to calculate a material's EOS, the equilibrium configurations corresponding to a set of isotropic pressures must be calculated using VCS-MD in damped mode. When C_{ij}s are necessary, following the first structural optimization step, a set of Lagrangian strains is applied to each equilibrium configuration and the resulting structures are relaxed again using fixed cell molecular dynamics (MD) in damped mode. This will produce a set of out of equilibrium internal stresses [20,21] that are used to compute the elastic constants at zero temperature. This approach does not take in consideration atomic vibrations. In order to correct the EOS and C_{ij} for zero point motion, or extend it to high temperatures, we must include the vibrational contribution to the free energy. Therefore, the vibrational density of states (vDOS) must be calculated for every configuration (strained or not) resulting from structural optimizations described above [13,17]. Among the several software packages available featuring this type of calculation, ***VLab*** has chosen the PWscf (*pw.x*) and phonon (*ph.x*) programs from the quantum ESPRESSO package [22].

MD, VCS-MD, or vDOS calculations of any configuration are accomplished by execution of one or more individual jobs. The process consists in a systematic workflow, where the output of a given task is used to compose the input of the next one. Traditionally, the analysis of each output, elaboration of the next input, job submission, and management of the large number of data files are performed manually. This is time consuming and very prone to human errors.

Two programs from the quantum ESPRESSO package, *pw.x*, for first principles structural and molecular dynamics calculations, and phonon (*ph.x*) for phonons calculations, are overwhelmingly responsible for the workload generated by thermal and elastic properties calculations in ***VLab***. MD, VCS-MD and eventually fixed configuration self-consistent calculation are performed using *pw.x*. The program *ph.x* calculates the dynamical matrix for a given atomic configuration at a given q-point. In most cases, input parameters are initial values of physical variables. Some, like cell vectors in MD calculations, pressure in VCS-MD, or cell vectors and q-point in phonons calculations, do not change during the execution cycle of the program, and are the ones frequently sampled in extensive studies.

Decoupled sampling of strains and points in Brillouin zone for phonons (q-points) is straightforward. Sampling pressure points, however, is more complicated. This happens because starting from a configuration very different from the equilibrium is inefficient, as structural optimization will take much longer to converge. The usual approach to minimize this problem is to optimize structures in increasing order of pressures where the structural parameters obtained as output of P_n are used as input for P_{n+1}. However, this approach introduces an undesirable coupling between pressure points preventing them from running concurrently.

Fortunately, in most cases, what changes most with pressure is the cell volume. This is the key to decouple pressure sampling. The calculation starts with a crude guess for the equation of state (EOS) that allows for a rough estimation of the cell volume at each pressure independently. Then a single self-consistent calculation, corresponding to just one MD step, is performed for each pressure. The virial pressures are calculated and used to obtain a better estimate for the EOS. This EOS, although still imprecise, is capable of producing configurations that are sufficiently near to equilibrium configuration to allow for a fast convergence of the subsequent VCS-MD stage. We named this preliminary stage as fast refine (FR).

2.1. *VLab* workflows

The general abstract workflow for C_{ij} calculation is as follows:

1. Start with a set of pressures $\{P_i\}$.
2. Crude guess for the parameters of Vinet EOS, V_0, K_0, K', and cell parameters for $T = 0$ K and $P = 0$ [23].

   $$P = 3K_0 \frac{1-f}{f^2} \exp\bigl[1.5(K'-1)(1-f)\bigr],$$

   with $f = (V/V_0)^{1/3}$, and V_0 being the cell volume at zero pressure. Initial guesses for the cell volume from atomic radii, and $K' = 4$ are rough but satisfactory. Bulk modulus can be guessed from any other material with similar composition and density. This produces a set of starting cell volumes, corresponding to the input pressure set.
3. Compute pressure P, with a single static configuration, for every volume generated in step 2 (decoupled runs).
4. Fit the Vinet EOS to the $P \times V$ data obtained from steps 2–3.
5. Use EOS from step 4 to calculate the new set of cell volumes.
6. Full VCS-MD structural optimization starting from structures obtained in step 4 at respective pressures (decoupled runs).
7. Fit the Vinet EOS to the $P \times V$ data from steps 5–6, producing a precise EOS.
8. Generate a set of strains $\{\epsilon_j\}$ and apply them to all optimized configurations obtained in step 6, generating $card(\{P_i\} \times \{\epsilon_j\})$ configurations. For sake of accuracy, at least two strained configurations must be generated for each longitudinal and off diagonal strains. Sometimes, only one configuration is sufficient for off-diagonal strains. If C_{ij} are going to be computed at high temperatures, combinations of $\{\epsilon_{ij}\}$ must be applied simultaneously to calculate the second order derivatives.
9. Optimize all internal degrees of freedom for all structures generated in step 8, using fixed cell MD ($card(\{P_i\} \times \{\epsilon_j\})$ decoupled tasks).
10. Calculate phonon spectra of structures obtained in steps 6 and 9. Usually phonons are calculated at several q-points for each structure. All tasks are decoupled. This step will generate $card(\{P_i\} \times \{\epsilon_j\} \times \{q_n\}) + card(\{P_i\} \times \{q_n\})$ tasks.
11. Gathers output files and runs post-processing of results from 11.

3. Metadata description

Definition of metadata is crucial in designing a system required to support concurrent execution in an heterogeneous distributed environment. ***VLab*** must keep metadata to track the execution of tasks and the locations of a plethora of input and output data files that lie scattered throughout several nodes.

VLab is composed of a collection of web services (WS) [24], each one with specific functions like workflow control, task dispatching, task execution, execution monitoring, and several visualization services. All the user visible services are interfaced through a portal that provides the abstraction of a single system. A precise and concise description of the ***VLab*** structure and metadata demand some preliminary definitions.

3.1. *Definitions*

project : The user-level abstraction for an extensive workflow. In order to execute a workflow, such as a new materials C_{ij} calculation or any of its subsets, the user has to define a new project. From the users perspective, it is composed of the main input file, and the pseudo-potential files. From the system perspective, it is represented by the project descriptor, and additional associated metadata.

project descriptor : The piece of metadata containing generic information about the whole workflow, such as its kind, the project receipt, the location of the main package and initial package.

stask : Short form of short task. A task that executes within the typical time out limit for a web transaction. In this case, the caller can wait until the called WS or internal class to complete the task and get the results back.

ltask : Short form of long task. A task that takes much longer to complete than the typical time out limit for a web transaction. In this case, the called service will acknowledge the transaction by sending back a receipt and the return of results under, task completion, is deferred for later time. In particular, the whole execution of the project, is an ltask.

outstanding ltask : Is an ltask that has been requested and is either waiting execution, is executing, or has been executed but its output has not been returned yet and the associated files have not yet been removed from the server that executed it.

receipt : A package of metadata, with well defined structure, describing an ltask that has been requested. The receipt contains all the information necessary to locate an ltask as well as all associated files, even if the task is composed by sub-tasks, and its files distributed throughout several servers. The receipt is a recursive data structure, containing a particular attribute, attachment, that is a list of receipts.

complaint : Any request relative to an outstanding ltask that is accompanied by the ltask receipt.

execution package : A directory with an ltasks input file, the eventually required pseudo-potential, and an empty subdirectory called tmp all the environment necessary to run an ltask. It is transferred as is to the execution node, and the respective ltask runs inside it.

initial package : A directory containing the main input file for an extensive workflow, a subdirectory called RunDirTemplate containing, the pseudo-potential files requested in the input and an empty sub-directory called tmp. It is used by the parameter sampling algorithms to generate the several execution packages.

image package : A directory containing a copy of the input file present in a respective execution package. At the end of the ltask associated with the respective execution package, or at user request, the ltask main output file is brought back to the image package for further analysis.

main package : A directory containing the same files and as the initial package, plus an Image package for every execution package. The main package is stored in a file system directly accessed by the project executor the web service responsible for controlling the execution of the workflow.

project manager : Is a component of the ***VLab*** user interface. It enables users to define new projects, save and re-open projects, set up preferences, create input data files, and upload pseudo-potentials to be stored in the initial package. It also creates the project descriptor when a new project is created.

3.2. Metadata management

VLab is a distributed system composed of several independent hosts. Consequently, it deals with tasks that are executed in hosts other than the one that controls the project. Moreover, it is implemented as a service oriented architecture (SOA). The several participating hosts provide functionality to the others by means of Web Services [24].

When a Web Service is executing an stask, it is acceptable that it keeps the session alive until it is able to send back the results. Many ltasks, however can run for several days or even weeks. In this case, it is convenient to run the ltasks unattended, under sole control of the remote host operating system. This way, the server can close the session as soon as the ltask is running or, otherwise, submitted to a queuing system. Before closing the session, however, the server must send back the metadata necessary to enable the client to retrieve the ltask results, monitor its status, or further interact with it.

Metadata management is based mostly on the receipt metaphor: In an analogous commercial operation, when a customer requests a service, it gets a receipt. This receipt contains the operations identification code, clerks name, branch number, etc. Any time the customer wants to check if service is complete, query the current status, or file a complaint in case of something wrong, the customer must show the receipt that has the necessary information to locate the order, make proof of payment, etc.

In ***VLab***, every time a server is requested to run an ltask, it gives back a receipt to the client. The receipt is a package of metadata containing all information necessary to retrieve any further data related to the outstanding ltask, like server identification, local path, PID (or JobID, if running in batch mode), service requested, receipt identification number, etc. Every time the client needs to retrieve any information, or request any further action (restarting the task, for example), it files a complaint by filling a designated field in the receipt with a code for that information or action, and send the receipt back to the server. Then, the server parses the complaint field, and take the necessary steps to acknowledge the complaint.

Under the project point of view, each phase is an ltask. Under the users point of view, the whole project is an ltask. These also must generate receipts. For this reason, the receipt was designed as a recursive data structure. The receipt has a field called attachment that is itself a list of receipts. When Project Executor submits the ltasks corresponding to a given phase, all ltask receipts are attached to the phase receipt, and the latter is, by its turn, attached to the project receipt. This way, the project receipt contains the receipts of every ltask in the project.

In order to achieve persistence, the project receipt is attached to the project descriptor and stored in a central database. The project executor also keeps a cached copy of the receipt.

4. System description

By its own nature, ***VLab*** must deal with a large number of outstanding ltasks that are related to each other. These ltasks are, in most cases *pw.x*, ph.x, or post-processing tools runs. These programs deal with two very distinctive kinds of files: (i) Small input, standard output, or intermediate text files. Due to its small to moderated sizes can be transferred multiple times between servers without generating significant network traffic. (ii) Huge binary files that contain wave functions, charge density, electrostatic potential, etc. Those files are intended to support resuming incomplete calculations. They are also used as input for the post-processing tools and for the ph.x. Usually, they can be re-created from data present in the standard output, avoiding large file transfers. Moreover, being FORTRAN binaries, their format is system dependent. Since the system is heterogeneous, the transferred files are useless in most cases. Re-creating the files is not the best strategy for charge density visualizations, since it is not reasonable to re-create those files in the visualization server. The solution in this case is to run the post-processing tool in the same server as the *pw.x* ltask has ran. Although post-processing runs are ltasks, they impose a much smaller workload than *pw.x* or ph.x, so they can always run in the same server as *pw.x* independently of current load conditions. Moreover, post-processing utilities outputs text files that although large, are much smaller than the binary files it uses as input and can be transferred without generating excessive network traffic or long delays.

VLab Workflows fall in one of six categories depending on extension:

(i) Simple execution of *pw.x* or ph.x codes. Those are usually preliminary or complementary calculations.
(ii) Pressure point sampling. Used mostly to determine a given materials Equation of State (static EOS). This is usually done in two phases, each one consisting of a dozen or so of *pw.x* runs. After the first phase, information must be collected from output of all *pw.x* runs to compute the input for the second phase.
(iii) Determination of a given materials Equation of State (finite temperature). Executed in three phases. The first two are identical to the case above. The third phase involves execution of the ph.x (several runs for each *pw.x* run from the previous phase). The *pw.x* temporary binary outputs are used as inputs for ph.x.
(iv) Determination of elastic constants, C_{ij} (static). Executed in three phases. The first two are identical to case (ii). The third phase executes several *pw.x* runs for each run from the second phase. This case also involves collection of output files from all *pw.x* runs of the previous phase.
(v) Determination of elastic constants (finite temperature). Besides the three phases involved in the previous case, it also involves several ph.x runs for each *pw.x* run in the second and third phases of case (iv).
(vi) Post-processing and visualization of results from (i)–(v). Although the categories (ii)–(v) generate a huge number of configurations, usually only a small number of those are selected for post-processing/visualization.

The general workflow of Section 2.1 can now be written in terms of concrete operations:

1. Using the portal, set up a general input containing a rough approximation to the EOS, and the pseudo-potential files.
2. Call the appropriate service to transform that general input in several *pw.x* inputs. Each input is intended for a single MD step calculation with estimated cell volumes for the corresponding pressure based in the EOS from step 1.
3. Run multiple distributed *pw.x* tasks using the input files from step 2 (Fast Refine).
4. Gather pressure and cell volume data from the distributed *pw.x* standard outputs generated in the previous step.
5. Fit the Vinet EOS.
6. Automatically modify the general input with new values for EOS parameters from step 5.
7. Call the appropriate service to transform that general input in several *pw.x* inputs for multi-step VCS-MD calculations using the new EOS. This will produce a new estimation for the initial cell volume.
8. Run multiple distributed *pw.x* tasks using the input files from step 7. We call this step Long Refine.
9. Execute again steps 4 and 5.
10. Gather cell vector data from the respective *pw.x* standard outputs.
11. Generate a user defined set of Lagrangian strains for each cell collected in step 10, and generate corresponding *pw.x* inputs.
12. Run multiple distributed *pw.x* tasks with fixed cell MD calculations from inputs generated in step 11.
13. Collect the *pw.x* standard outputs from step 12, and extract: a) the final stresses; b) final positions.
14. With the a user defined strains and stresses from step 13a, calculates a table of static elastic constants.
15. With the final cells from steps 10 and 11, atomic positions from 13, and user defined q-points, generate input files for the phonon jobs for every configuration resulting from steps 8 and 12. A phonon job comprises the execution of the following successive steps: (i) a *pw.x* run to perform a scf calculation of the final structure from steps 8 or 12; (ii) non-scf calculation if the q-point is not the gamma point; (iii) a ph.x run. The scf calculation is necessary to recreate the binary files used by the subsequent steps. All three calculations are carried in the same server.
16. Run multiple distributed phonon jobs from input files generated in step 15.
17. Collect all the dynamical matrices calculated by the ph.x runs in step 16 and calculate the constant force matrix for each configuration from steps 8 and 12. Then calculate the VDOS for those configurations.
18. Using the VDOS from step 17, calculate the high temperature EOS, C_{ij} and other thermal properties.

Fig. 1 shows a graphical representation of this workflow.

Step 1 is implemented in the portal. All remaining steps are executed by web services. Steps 3, 7, 11, and 13 involve execution of ltasks, and are intermediated by steps that recollect data and execute stasks. Extensive scattering/gathering of information is performed inside major blocks composed by steps 2–5, 6–8, 9–13a, 13b–14. These blocks are successions of the following generic three steps sequence: (i) Preparation of a set of inputs for ltasks, like *pw.x* or ph.x runs, and packing with appropriate companion files forming an execution package; (ii) Distribution of the execution package throughout the back-end computing nodes for execution; (iii) Gathering results for analysis and set up of parameters to iterate steps (i)–(iii).

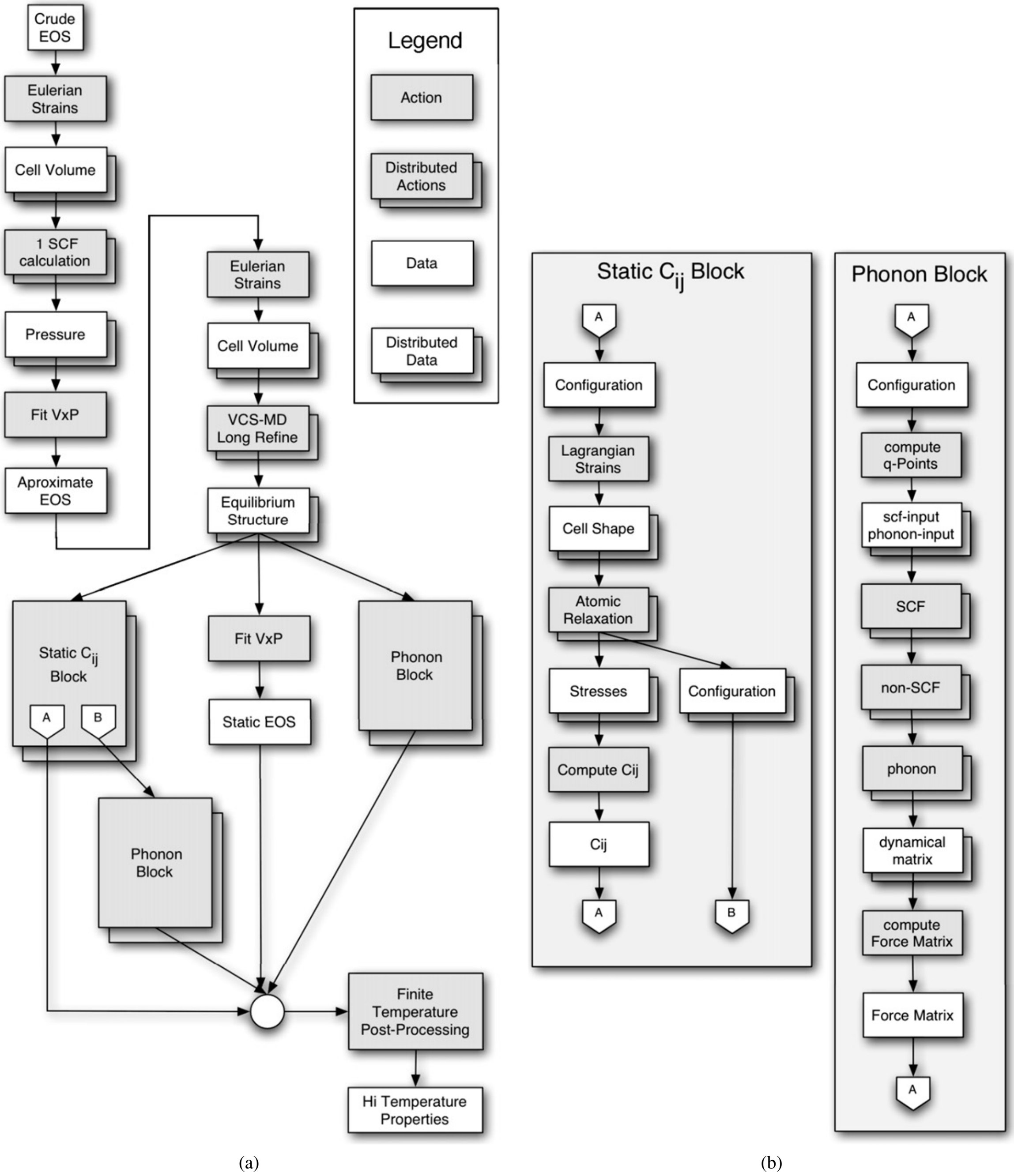

Fig. 1. Graphical representation of the workflow described in the text. (a) Main diagram showing detailed view of the EOS calculation. C_{ij} calculation and phonon calculations appear as black boxes. (b) Detailed view of C_{ij} and phonon boxes. This depiction shows clearly the nested workflow level parallelism existing in the C_{ij} and phonon calculations.

Moreover, in the overall process, input data comes primarily from the user and results must be, ultimately, sent back to the user.

The system is strongly oriented towards the problem. In this case, a usage-oriented view [25] is more appropriate to describe its design. It is also more effective as a mean of elucidating the several design questions that appear during the solution devise phase of development. For these reasons, a usage-oriented view was primarily used for development. It will also be used throughout the rest of this paper to describe the architecture related aspects of ***VLab*** SOA and metadata.

In order to better accomplish the three steps sequence discussed above, services are structured in a tree, with compute services at the leaves and main control functions at the root. A key characteristic is that our data flows are highly predictable and repetitive. This is in contrast with the LEAD project [26], for example, that aims an implementation fully compliant with the Open Grid Service Architecture (OGSA) specification [27]. In LEAD, data can come from a variety of sources in an unpredictable way, and must be sent to a multitude of destinations according to the particular application workflow that is collecting the data. In ***VLab***, the scenario is much simpler. Most

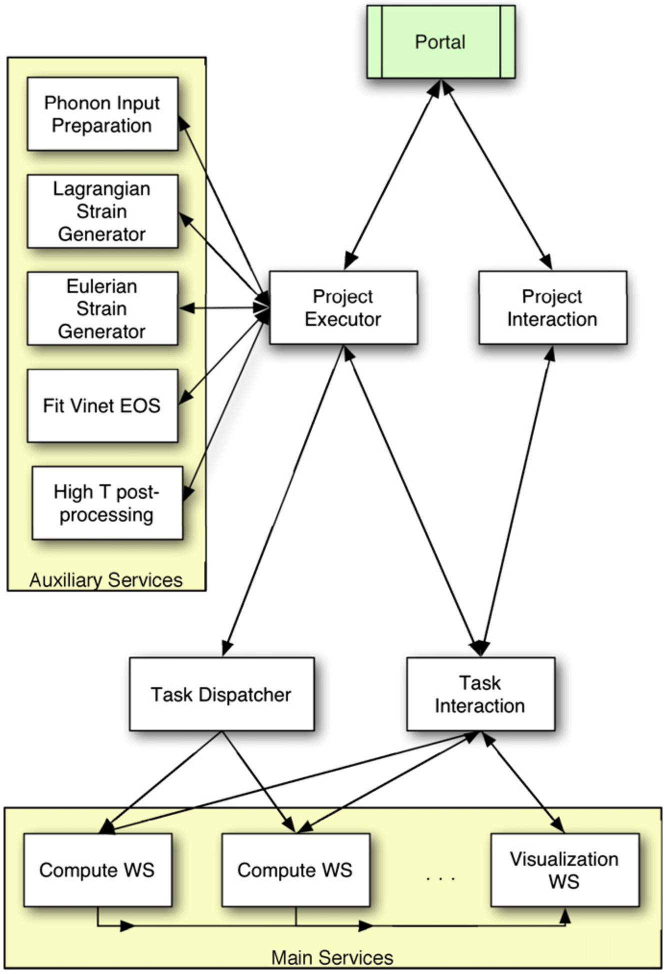

Fig. 2. VLabs tree structure.

times a service is requested, the client sends input data to the server and receive back the service output data. Thanks to this reason, full OGSA compliance is not strictly necessary, and the requirement for a strong grid service layer can be relaxed. ***VLab*** presently implements only an essential subset of OGSA functionality. An increasing level of compliance with OGSA is slated for future versions. This choice allows for a much faster development cycle with limited resources, and the resulting system is still capable to fulfill its goals. The overall structure is shown in Fig. 2.

The main services are in the bottom box. Its functions are to start ltasks in the underlying operating system of the visualization or compute server and send results back to the client. Compute services must also have the ability to forward results from the post-processing tool directly to the visualization server to avoid unnecessary network traffic. This is easily done within the receipt protocol. When the visualization post-processing starts it sends back a receipt containing, among other things, the URL of the visualization data file that will be generated. When the completion of the post-processing ltask is detected, the visualization server is notified and downloads the data file from the compute server.

The computing nodes are interfaced to the rest of the system by the Compute WS web services. These are purely transactional stateless web services. Many ltasks can run for several days or even weeks. Therefore, it is convenient to run the ltasks unattended, under sole control of the host operating system. When executing a request, these servers close the session as soon as the requested ltask is running, or otherwise submitted to a queuing system. Before closing the session, however, the server sends back the receipt to enable the client to retrieve the results, monitor execution status, or further interact with the running task.

Input files are staged to the Compute WS in the form of a compressed file called execution package. It contains a directory with one or more *pw.x* or ph.x input files, the pseudopotential files requested in the input file and an empty subdirectory called tmp all the necessary environment to run the task. The execution package is transferred to the execution node as is, and the respective task runs inside it. Output files, like the standard *pw.x* output, or the dynamical matrix produced by the ph.x code will also be stored in that directory. After execution, or under user request, output files, are staged back to the main package. The staging back process always occurs under request of the Project Executor through Task Interaction using the complaint mechanism.

The topmost service is the Project Executor. Its function is to control the execution of the workflow at higher level. It calls the auxiliary services that execute several short tasks (stasks), send ltasks for execution, gather results to be fed in the auxiliary services, process output files from one phase to produce input files for the next phase, controls the sequence of phases, and take automated decisions to control the overall workflow execution.

On entry, the project executor receives the project descriptor and the basic package. It stores in a database the project receipt and the project descriptor, creates the main package aimed to keep images of the ltasks input and output files. Consistency with the files in the backend is guaranteed at the end of the ltask execution and at every user interaction.

The Task Dispatcher is responsible for electing back-end nodes and dispatching ltasks for execution. It gathers information on current load conditions in each compute server prior to request execution of heavier ltasks (*pw.x* and ph.x).

Task Interaction is currently implemented as a library shared by both Project Executor and Project Interaction. Its function is to retrieve data and task status from the compute nodes and keep consistency of the main package. It also supports the features necessary for user intervention in running projects, and provides interfaces for accessing the semaphores used to control shared access to task files.

There are also several auxiliary services. Their function is to execute stasks that are usually intermediate between major phases in the workflow. They are characterized by fast execution cycles and return results almost instantly. Consequently they do not need to generate any persistent metadata. The Eulerian and Lagrangian strain generators are auxiliary services responsible for generating the several input files corresponding to parameter sampling in pressure and cell parameters. There are also services for fitting the Vinet EOS, preparing ph.x input files, and post-processing of ph.x output files (labeled as High T post-processing in Fig. 2).

Project interaction is a web service that supports the project monitoring user level functionality in the portal. This module allows for user interactions with projects and all tasks that compose it. In ***VLab***, collaboration is established by means of shared access to projects, including file handling, and tasks management. Users can intervene in a project by modifying input files, and canceling or restarting tasks. Project interaction is responsible for supporting this kind of shared access by precluding race conditions that could arise from it, and potentially compromise data integrity.

The portal is responsible for providing a consolidated single point interface to the main services. It provides the user level abstraction of a single system, while providing access to the underlying resources. It aggregates the interfaces of Project Interactions, Project Executor, Visualization services, and analyses tools. It also allows for project management, input file preparation and uploading pseudo-potential files.

4.1. Metadata for collaborative use support

VLab is intended to support collaborative work through shared access to projects that are in preparation, running, or complete with results under analysis. To support this kind of functionality, Project Manager and Project Monitor must grant, to several users and to each other, simultaneous write access to shared files, and allow interactions with outstanding ltasks. This is the typical scenario where race conditions can arise. Suppose, for example, that a given task is taking too long to converge. Its input file must be edited and the task subsequently restarted. A given user opens the input file for editing and, before saving the changes, another user does the same. Next, both users save the file and restart the task. The result will be unpredictable and probably wrong. The system must preclude race conditions and ensure that critical operations are performed consistently. Race conditions can arise even when only one user is trying to intervene in a project. The Project Monitor, working on behalf of the user, and the Project Executor can still rush for shared resources.

In order to support shared access to resources, the metadata must include a set of semaphores. One semaphore is included in the project descriptor. Each receipt has also its owns semaphore. Consequently there are semaphore metadata related to each ltask, including the project itself. Each of these semaphores, controls the access to the respective task and task data.

Since the Project Manager works with a cached copy of the project receipt, it is also important to keep the metadata in the central database and the cached copy consistent. Therefore, in addition to the semaphores, a counter is included in each receipt and in the project descriptor. The counters count the number of modifications the receipt or project descriptor has previously undergone. A simple comparison of the counter in the database against the cached copy is enough to check the consistency of the cached receipt.

The access to semaphores is, by its turn, guaranteed to be sequential by using the data locking mechanisms available in the database server.

5. Metadata implementation

5.1. Receipt

The receipt data structure is implemented as the java class shown in Listing 1a–1b. It has several attributes of type List, that implements a list of objects of variable length, as described in Ref. [28]. The attributes attachment, acknowledgment, content, and complaint are lists. Although it is not possible to restrict the base type for a list, those attributes are consistently used with a single base type each. This type is identified in the comments in Listing 1-b. The base type for acknowledgment, content, and complaint is bdata. The base type for attachment is receipt, which makes the receipt a recursive data structure. The class bdata is defined in Listing 1-a.

The bdata class contains data in the format (<code>, <keyword>, <value>), where code is an integer, keyword and value are strings. If the values corresponding to a given keyword are typically numerical, the numbers are converted to a string representation for storing purpose. Code and keyword semantically redundant. Consequently, we will hereafter mention only keyword and value, leaving code implicit. This redundancy is very helpful in developmental stage because it eases debugging by allowing consistency checking.

One important requirement for the receipt structure is flexibility. Receipts have to hold metadata relative to very different kinds of ltasks: *pw.x*, phonon jobs, several post-processing tasks, phases and full projects. This flexibility is provided by lists of elements of (<keyword>,<value>) format. Lists can hold an undetermined number of elements, and the format (<keyword>,<value>) with keyword and value stored as strings allows for the choice of metadata that best suites every case.

The attributes mySemaphore and countChanges have been previously discussed. If mySemaphore is false, the ltask is unlocked and its data files, execution status and the receipt itself can be modified. If it is true, the ltask is locked, and only the process that has the lock can modify the ltask status or related data. If mySemaphore is true, the attribute content also contains the identification of the client that have set the semaphore. The attribute countChanges is used to keep track of the number of modifications the receipt has undergone since it was generated. It is used to check consistency between the receipt cached copy in Project Executor and the persistent copy stored in the metadata database.

The attributes kind and code are used to identify the kind of ltask (backend task, phase, project) that generated the receipt and the operations executed (VCS-MD, phonon job, etc.) respectively. Values for projectid and projectname are set by Project Executor, and primarily intended to phase and project receipts. Compute WS also receive these parameters when task executions are requested since they are used to compose the namespace for the execution directories. Compute WS then incorporates the parameters in the ltasks receipts. This redundancy is convenient because it makes unnecessary to parse a recursive data structure every time the parameters are needed.

```
package org.vlab.services;
import java.util.*;
public class bdata
  implements java.io.Serializable{
    private int    code;
    private String keyword;
    private String svalue;
    //Setter
    public void setCode(int newCode){
        this.code = newCode;
    }
    public void setKeyword(String newKeyword){
        this.keyword = newKeyword;
    }
    public void setSvalue(String newSvalue){
        this.svalue = newSvalue;
    }
    //Getter
    public int getCode(){
        return this.code;
    }
    public String getKeyword(){
        return this.keyword;
    }
    public String getSvalue(){
        return this.svalue;
    }
    public bdata(){}
}

package org.vlab.services;
import java.util.*;
public class receipt {
    private Boolean   mySemaphore;
    private int       countChanges;
    private int       kind;
    private int       code;
    private String    projectid;
    private String    projectname;
    private List      content;
    private List      complaint;
    private List      acknowledgment;
    private List      attachment;
    //setters
    public void setMySemaphore(
      Boolean newMySemaphore){
        this.MySemaphore = newMySemaphore;
    }
    public void setCountChanges(
      int newCountChanges){
        this.CountChanges = newCountChanges;
    }
    public void setKind(int newKind){
        this.kind = newKind;
    }
    public void setCode(int newCode){
        this.code = newCode;
    }
    public void setProjectid(
      String newProjectid){
        this.projectid = newProjectid;
    }
    public void setProjectname(
      String newProjectname){
        this.projectname = newProjectname;
    }
    public void setContent(List newContent){
        this.content = newContent;
    }
    public void setComplaint(
      List newComplaint){
        this.complaint = newComplaint;
    }
    public void setAcknowledgment(
      List newAcknowledgment){
        this.acknowledgment = newAcknowledgment;
    }
    public void setAttachment(
      List newAttachment){
        this.attachment = newAttachment;
    }
    //Getters
    public boolean getMySemaphore(){
        return this.MySemaphore;
    }
    public int getCountChanges(){
        return this.CountChanges;
    }
    public String getKind(){
        return this.kind;
    }
    public String getCode(){
        return this.code;
    }
    public String getProjectid(){
        return this.projectid;
    }
    public String getProjectname(){
        return this.projectname;
    }
    public List getContent(){
        return this.content;
    }
    public List getComplaint(){
        return this.complaint;
    }
    public List getAcknowledgment(){
        return this.acknowledgment;
    }
    public List getAttachment(){
        return this.attachment;
    }
    public receipt(){}
}
```

Listing 1. Class ***receipt***, and the auxiliary class ***bdata***.

Receipts for phases have empty contents list. Receipts for Compute WS tasks and projects include the following data in the contents list:

(a) – Compute WS tasks:
- PathExec : The path where input and main output files are stored in the execution server. The innermost directory in the path is the execution package.
- IPAddress : The IP address where the Compute WS server responsible for executing the task is running.
- PortNumber : The TCP port number used by Compute WS.
- HostName : Host name of the machine running Compute WS.
- ProcessID/JobID : If the ltask has been forked directly by Compute WS, *keyword* will be "ProcessID" and *Value* contains the PID of the ltask. The *keyword* will be "JobID", and *Value* will contain the job ID of the ltask if Compute WS submitted it to a local batch queue.
- StartTime : The time at the execution server location when the task started.
- ProcessStatus : The current status of the task.
- PackageName : The name of the Execution Package. It is the same as the innermost directory in the path indicated by *PathExec*.
- EndTime : The time at the execution server location when the task ended.
- ChDensGenerated : Indicates if all data necessary to run the post-processing for visualization of charge density has already been generated.

(b) – Projects:
- MainPackageDir : The directory name of the main package, including complete path.
- IPAddress : The IP address where the Project Executor server responsible for executing the project is running.
- PortNumber : The TCP port number used by the Project Executor web service.
- HostName : Host name of the machine running the Project Executor.

The complaint list contains requests relative to the outstanding ltask. When a client needs anything regarded to the ltask, it empties the *complaint* and *acknowledgment* lists, in order to erase the effects of any previous complaint, fills the *keyword* and eventually the respective *value* for each list element that compose the complaint and send the receipt to the server that issued the receipt. If the answer to the complaint is short, like an status update, or a path to a file, it comes embodied in the *acknowledgment* list. If the answer is long, like sending to the client the main *pw.x* output file, the answer will come through the main web service interface, and the *acknowledgment* will contain an indication of that. The *acknowledgment* will also contain a proper indication if the server failed to fulfill the request.

The *attachment* list is intended to be used primarily in phase and project receipts. In phase receipts, the *attachment* list holds Compute WS receipts. In project receipts, the *attachment* list holds phase receipts. This structure is shown in Fig. 3. Compute WS receipts contain empty *attachment* lists.

As the hierarchic structure contains only three levels, using a more general recursive data structure could be thought as exaggerated. We could very well define a different receipt class to be used in every level. However, the methodology for this class of problem is constantly evolving, demanding flexibility in the implementation of project management. Our choice is aimed to preserve that flexibility.

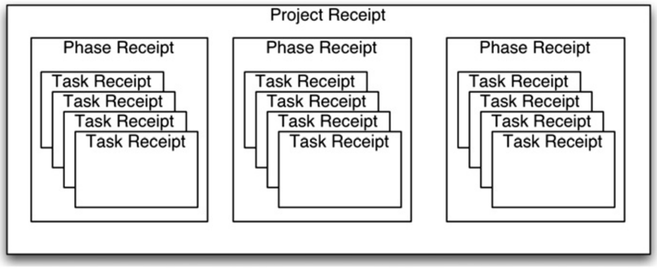

Fig. 3. Hierarchic structure of a project receipt.

In order to transfer the receipts from server to server and store it in databases, we need to convert the receipt to some format other than a java class. ***VLab*** adopts XML for this purpose. The castor library [29] provides the means to store the contents of a java class attributes as an XML file. The structure of the XML file matches the hierarchical structure of the original java class with each XML tag associated to the corresponding class attribute. The XML file can then be transfered, or stored in a database for sake of persistence. Castor also provides the means to convert it back to java for further processing.

5.2. *Project descriptor*

The project descriptor is stored in a database that provides data persistence for the project receipt, including all its attachments, and other important metadata. This paper is mainly concerned with metadata related to distributed execution and collaborative monitoring and steering of projects in a distributed environment. For this reason, we will only describe here the fields actually used for that purpose. A broader description of the project descriptor as well as the entire ***VLab*** system will be subject for another paper.

The diagram of the project descriptor is shown in Fig. 4. It is composed of a central table, called TBPROJECTDESCRIPTOR, which is related to five other tables. The tables TBCURRENTPHASE, TBPROJECTTYPE, and TBSTATUSCURRENTPHASE are all part of the project descriptor abstract data structure. In contrast, TBUSER and TBGROUP are used to support collaborative project access, but are not part of the project descriptor. They are shown, however, because of their relations with TBPROJECTDESCRIPTOR and each other.

It is noteworthy that a field in TBPROJECTDESCRIPTOR stores the authoritative copy of the project receipt. Another field, tablesemaphore, is the main semaphore for all project metadata. Description of relevant fields is as follows:

TBPROJECTDESCRIPTOR

Each line of this table contains data relative to one project. A new entry is created every time the user creates a new project. The fields that contain metadata relevant to this paper are the following:

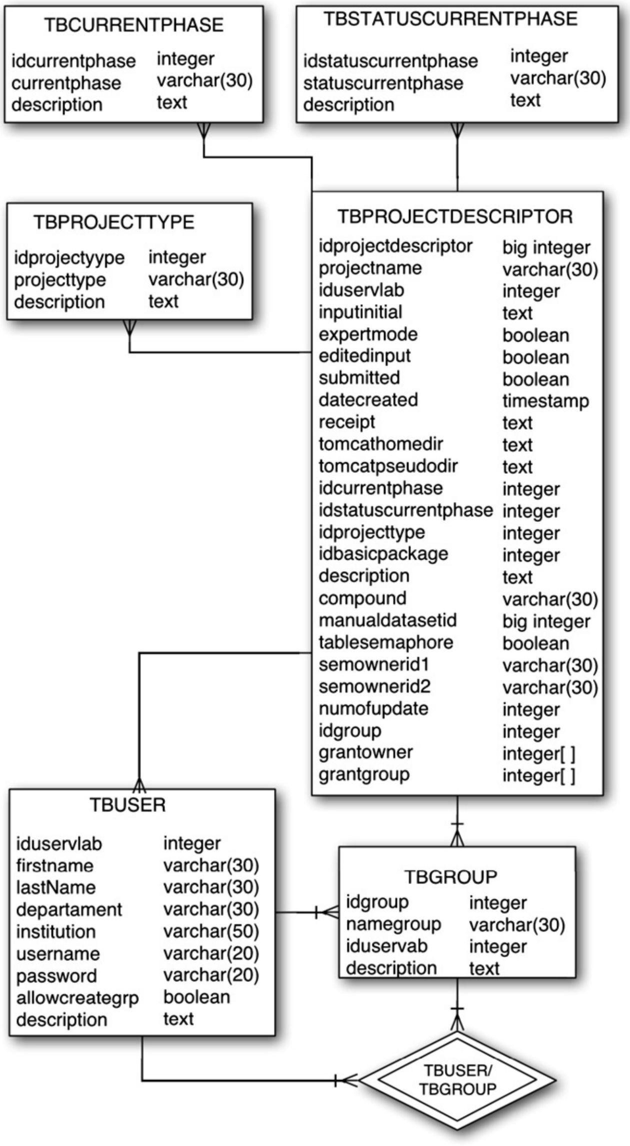

Fig. 4. Project descriptor is composed of four tables, TBPROJECTDESCRIPTOR, TBCURRENTPHASE, TBPROJECTTYPE, and TBSTATUSCURRENTPHASE plus the relation between TBGROUP and TBUSER.

idprojectdescriptor::integer : Primary key. Project identification number. It is uniquely generated by the system, and is used as an unmistakable identification of the project.

projectname::varchar(30) : Project Name, defined by the user. Although the system uses preferably the idprojectdescriptor, projectname is used in interactions with the user.

iduservlab::integer : Project owner user identification number. It is a foreign key for the table TBUSER. This field is used in conjunction with idgroup, grantowner, grantgroup, and TBUSER table to establish shared access privileges for collaborative purposes.

idgroup::integer : Identification number for the group attribute of the project. It is a foreign key for the table TBGROUP.

submitted::boolean : Its true if the project has already been submitted.

grantowner::integer[ ] : Permissions for the project owner. The first three positions in the array represent read (r), write (w), and execute (x). A number different from zero means that permission is granted. ***VLab*** uses a metaphor similar to UNIX to handle access privileges of the users who collaborate in a project. There are three privileges: read, write and execute. Read means the user can only execute operations that do not interfere in the normal execution of automated workflows or alter its input and output files. The user can execute tasks like checking execution status, see output files, or perform visualizations. With write privilege the user can edit input files such as the initial input, or change task inputs in order to steer them. However, it is also necessary execute privilege to actually steer tasks or change task status. The default permissions are rwx for the owner, and r—for the group.

grantgroup::integer[ ] : Permissions for users participating in the group defined by idgroup.

receipt::text : Current receipt for the project. This is the authoritative copy of the receipt. Any process must check consistency with this copy before using any cached receipt. Before changing the receipt, the status of any related ltask, or any data related to these ltasks, the process must first acquire exclusive access to this copy of the receipt and set the appropriate semaphores.

idcurrentphase::integer : This field is a foreign key for the table TBCURRENTPHASE. It contains the identification of the particular phase that is currently running.

idcurrentphasestatus::integer : Foreign key for the table tbstatuscurrentphase. It indicates the current status of the current phase.

idprojecttype::integer : Foreign key for the table tbprojecttype. It contains the project type defined by the user.

idbasicpackage::integer : Foreign key for the table tbbasicpackage (not shown in the figure), which actually contains a back-up copy of the basic package in compressed format. The copy is crated by project executor when it starts execution of the project, and is updated every time its content undergoes relevant changes. A change is considered relevant if the new state of the basic package cannot be easily restored from the backend nodes or the restore can not be guaranteed for accuracy.

tablesemaphore::boolean : Semaphore that controls access to the project, which is initially set to false. Before making any change to the project descriptor or the included project receipt, the client must first acquire exclusive access by setting the semaphore to true in an operation protected by a lock. Once the semaphore is set, no other client can make changes to the project descriptor.

semownerid1::varchar(30) : If the semaphore is set, it contains the identification of the client system that requested the semaphore.

semownerid2::varchar(30) : If the semaphore is set, it contains the session id of the client that requested the semaphore.

numofupdate::integer : Number of changes made to the project receipt since it was created. This allows for consistency checking of the cached copies of the receipt.

TBPROJECTTYPE

This table contains the types of project supported by the system. Each line contains the specifications for one project type.

IdProjectType::integer : Primary key. The project type identification number.

ProjectType::varchar(20) : The project type represented as a string.

Description::text : Is a simple description for every project type created.

TBCURRENTPHASE

This table contains all phases that projects can undergo. Each line contains the specifications for one phase.

idcurrentphase::integer : Primary key. Phase identification number.

currentphase::varchar(30) : Name of the phase represented as a string.

Description::text : Is a simple description of the phase, used for annotation purposes.

TBCURRENTPHASESTATUS

This table contains all the possible status that any phase can assume. Each line contains the specifications for one possible status.

idstatuscurrentphase::integer : Primary key. Phase status identification number.

statuscurrentphase::varchar(30) : Phase status.

TBUSER

Table containing data about the users, including authentication information. In this table, each line contains data for one user.

IdUserVlab::integer : Primary key. User identification number.

UserName::varchar(20) : User log in name.

Password::varchar(20) : Password used to log into the system. For sake of security, the password is stored in encrypted form.

allowcreategrp::boolean : If true, the user has the ability to create new groups and add users to them.

TBGROUP

This table contains data about the groups. Each line contains data for one group.

idgroup::integer : Primary key. Group identification number.

namegroup::varchar(30) : Group name.

iduservlab::integer : User identification number of the group creator. It is a foreign key for the table TBUSER.

TBUSER/TBGROUP

Relation that establishes which users are included in each group.

6. Summary

We have explained the most extensive workflow planned for ***VLab***. We have shown in a usage-oriented view the SOA necessary to accomplish that workflow and explained how the receipts metadata metaphor works and how the receipt data structure naturally arise from the SOA. A key aspect of ***VLab*** is the support for collaborative work in projects. Substantial discussion was devoted to the issues involved in enabling collaborative user actions on projects executing in a concurrent distributed system. We have shown how those problems can be solved using standard techniques from concurrent systems with a little help from the data locking mechanisms available in most database servers. In addition, complementary metadata used to handle distributed execution were also described. This includes the metadata necessary to implement the users and group access permissions metaphor, borrowed from Unix, also necessary to enable collaborative work.

Acknowledgements

Research was supported by NSF grant ITR-0426757 (VLab). Minnesota Supercomputing Institute provided the main computational facilities.

References

[1] See http://VLab.msi.umn.edu/projects/ITResearch.shtml.

[2] P.V. Coveney, G. De Fabritiis, M.J. Harvey, S.M. Pickles, A.R. Porter, Coupled applications on distributed resources, Comp. Phys. Comm. 175 (2006) 389.

[3] P.V. Coveney, R.S. Saksena, S.J. Zasada, M. McKeown, S. Pickles, The application hosting environment: lightweight middleware for grid-based computational science, Comput. Phys. Comm. 176 (2007) 406.

[4] D.A. Yuen, B.J. Kadlec, E.F. Bollig, W. Dzwinel, Z.A. Garbow, C.R.S. da Silva, Clustering and visualization of earthquake data in a grid environment, Visual Geosci. 10 (2005) 1.

[5] M. Aktas, G. Aydin, A. Donnellan, G. Fox, R. Granat, L. Grant, G. Lyzenga, D. McLeod, S. Pallickara, J. Parker, M. Pierce, J. Rundle, A. Sayar, T. Tullis, iSERVO: Implementing the international solid earth research virtual observatory by integrating computational grid and geographical information web services, Pure Appl. Geophys. 163 (2006) 2281.

[6] D. Gannon, B. Plale, M. Christie, L. Fang, Y. Huang, S. Jensen, G. Kandaswamy, S. Marru, S.F.L. Pallickara, S. Shirasuna, Y. Simmhan, A.

Slominski, Y.M. Sun, Service oriented architectures for science gateways on grid systems, in: Lecture Notes in Computer Science, vol. 3826, 2005 p. 21.

[7] B.K. Godwal, S.K. Sikka, R. Chidambaram, Equation of state theories of condensed matter up to about 10 TPa, Phys. Rep. 102 (1983) 121.

[8] V.N. Zharkov, V.A. Kalinin, Equation of State for Solids at High Pressures and Temperatures, Consultants Bureau-Plenum, New York.

[9] G. Masters, G. Laske, H. Bolton, A.M. Dziewonski, Earths deep interior: from mineral physics and tomography from atomic to the global scale, in: S. Karato, A.M. Forte, R.C. Liebermann, G. Masters, L. Stixrude, (Eds.), Geophysical Monograph Series, vol. 117, 63 AGU, Washington, DC, USA, 2000.

[10] M. Ishii, J. Tromp, Normal-mode and free-air gravity constraints on lateral variations in velocity and density of Earth's mantle, Science 285 (1999) 1231.

[11] A.M. Dziewonski, D.L. Anderson, Preliminary reference Earth model, Phys. Earth Planet. Int. 25 (1981) 297.

[12] C.R.S. da Silva, R.M. Wentzcovitch, A. Patel, G.D. Price, S.I. Karato, The composition and geotherm of the lower mantle: Constraints from the elasticity of silicate Perovskite, Phys. Earth Planet. Int. 118 (2000) 103.

[13] R.M. Wentzcovitch, B.B. Karki, M. Cococcioni, S. de Gironcoli, Thermoelastic properties of $MgSiO_3$ Perovskite: Insights on the nature of the Earths lower mantle, Phys. Rev. Lett. 92 (2004) 018501. Phys. Rev. Focus story, What is Down There?, http://focus.aps.org/story/v13/st1, January 9 (2004).

[14] R.M. Wentzcovitch, Invariant molecular dynamics approach to structural phase transitions, Phys. Rev. B1 44 (1991) 2358.

[15] R.M. Wentzcovitch, J.L. Martins, First principles molecular dynamics of Li: Test of a new algorithm, Sol. Stat. Comm. 78 (1991) 831.

[16] R.M. Wentzcovitch, J.L. Martins, G.D. Price, Ab initio molecular dynamics with variable cell shape: application to $MgSiO_3$, Phys. Rev. Lett. 70 (1993) 3947.

[17] B.B. Karki, R.M. Wentzcovitch, S. de Gironcoli, S. Baroni, First principles determination elastic anisotropy and wave velocities of MgO at lower mantle conditions, Science 286 (1999) 1705.

[18] P. Hohenberg, W. Kohn, Inhomogeneous electron gas, Phys. Rev. 136 (1964) B864.

[19] W. Kohn, L. Sham, Self-consistent equations including exchange and correlation effects, Phys. Rev. 140 (1965) A1133.

[20] O.H. Nielsen, R.M. Martin, First-principles calculation of stress, Phys. Rev. Lett. 50 (1983) 697.

[21] O.H. Nielsen, R.M. Martin, Quantum-mechanical theory of stress and force, Phys. Rev. B 32 (1985) 3780.

[22] See http://www.pwscf.org.

[23] P. Vinet, J. Ferrante, J.R. Smith, J.H. Ross, A universal equation of state for solids, J. Phys. C 19 (1986) 467.

[24] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, Web services description language (WSDL), Version 1.1, March 15, 2000; see http://www.w3.org/TR/wsdl.

[25] G. Laures, Are service-oriented architectures the panacea for a high-availability challenge?—A position statement, in: Lecture Notes in Computer Science, vol. 3694, 2005, p. 102.

[26] K.K. Droegemeier, K. Brewster, M. Xue, et al., Service-oriented environments for dynamically, interacting with mesoscale weather, Comput. Sci. Engrn. 7 (6) (2005) 12.

[27] I. Foster, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, H. Kishimoto, F. Maciel, A. Savva, F. Seibenlist, R. Subramaniam, J. Treadwell, J. Von Reich, The open grid service architecture, V. 1.0, www.ggf.org/ggf_docs_final.htm, GFD. 30, July 2004.

[28] See http://java.sun.com/j2se/1.4.2/docs/api/java/util/List.html.

[29] See http://www.castor.org.