Using Services and Service Compositions to Enable the Distributed Execution of Legacy Simulation Applications

Mirko Sonntag[1], Sven Hotta[1], Dimka Karastoyanova[1], David Molnar[2], and Siegfried Schmauder[2]

[1] Institute of Architecture of Application Systems, University of Stuttgart, Universitaetsstrasse 38, 70569 Stuttgart, Germany
{sonntag,hotta,karastoyanova}@iaas.uni-stuttgart.de
[2] Institute of Materials Testing, Materials Science and Strength of Materials, University of Stuttgart, Pfaffenwaldring 32, 70569 Stuttgart
{david.molnar,siegfried.schmauder}@imwf.uni-stuttgart.de

Abstract. In the field of natural and engineering science, computer simulations play an increasingly important role to explain or predict phenomena of the real world. Although the software landscape is crucial to support scientists in their every day work, we recognized during our work with scientific institutes that many simulation programs can be considered legacy monolithic applications. They are developed without adhering to known software engineering guidelines, lack an acceptable software ergonomics, run sequentially on single workstations and require tedious manual tasks. We are convinced that SOA concepts and the service composition technology can help to improve this situation. In this paper we report on the results of our work on the service- and service composition-based re-engineering of a legacy scientific application for the simulation of the ageing process in copper-alloyed. The underlying general concept for a distributed, service-based simulation infrastructure is also applicable to other scenarios. Core of the infrastructure is a resource manager that steers server work load and handles simulation data.

Keywords: Service compositions, simulation workflows, distributed simulations, BPEL, Web services.

1 Introduction

The importance of computer simulations increases steadily. Nowadays many scientific institutes utilize computer simulations for their research. Due to achievements in information technology it is possible to use more and more complex and hence realistic simulation models. Nevertheless, there is still potential to improve existing solutions. In collaborations with scientific institutes in the scope of our research project we perceived that many simulation applications are based on legacy software that was developed over years and is still in development process. Many authors contributed to the software and may already have left the institute or organization. Usually, there is no time, money or knowledge to re-implement the tools in a modern programming language. The software is simply enhanced with new features. We experienced that there are many simulation tools in use that do not benefit from multi-core CPUs, distributed computing, Grid computing or computer clusters. The programs often cannot deal with parallel invocations, e.g. because they do not organize the result files appropriately. The simulation applications are monolithic or consist of only a few applications with simple, usually command line-based interfaces. It is even common that simulations are programmed and compiled into an executable (e.g. simulations based on Dune, http://www.dune-project.org). In this case, the simulation parameters are hard-coded and can only be changed by programming and re-compilation. There are a lot of manual tasks for scientists, such as copying result files between the participating applications, starting these applications, or merging results. These and other problems leave room for improvements of existing scientific simulation applications.

In this paper we present a conceptual architecture for a distributed simulation environment based on SOA and service compositions. In the last 5 to 10 years much research has been done to apply workflows for scientific applications (e.g. [1, 2, 3, 4]). We are convinced that workflows and service compositions possess great potential to improve the tool support for many scientists. These are the tools scientists work with every day. There is a need to automate and optimize simulation processes, to exploit recent developments in hard- and software, and to improve user-friendliness and flexibility of simulation applications. This can be done if modern IT and software engineering technologies and techniques are used. At the same time, scientists do not want to re-write existing code or re-implement applications for simulation tasks. The proposed concept addresses these and other requirements.

The main contributions of the paper are follows: (1) based on a scenario for the simulation of solid bodies and on our experience on software projects with scientists we have conceived a concept for a service-oriented simulation infrastructure. Major part of the infrastructure is a resource manager that steers work distribution between the scientific services and that handles simulation data. (2) we have implemented the concept for the simulation of solid bodies with Web services (WS) and BPEL [5]. Where possible we adopt existing concepts, e.g. from Grid computing.

The paper is structured as follows. Section 2 presents the real use case for the simulation of solids. In Section 3 we discuss related work. In Section 4 we describe the service composition-based simulation infrastructure using the example of Section 3. Section 6 closes the paper with a conclusion.

2 Related Work

Using service compositions to orchestrate scientific applications is not new. E.g. for BPEL the applicability in the scientific domain is shown in [1, 6, 7], for YAWL in [8]. To the best of our knowledge no service-oriented scientific workflow system makes use of a resource manager as middleware for load balancing and simulation context storage.

The Message Passing Interface (MPI) [9] is a specification of a programming model that allows parallel computing in Grids or computer clusters by exchanging messages, accessing remote memory or by parallel I/O. MPI provides a set of operations to implement the communication between applications. MPI-based programs mix process (or communication) logic and domain logic which is the main difference to our approach with workflows/service compositions. This minimizes the communication overhead but increases the programming effort for scientists. Our service-based solution relieves scientists from the knowledge about other applications when implementing their modules/services, increases reusability of their applications, allows monitoring of the simulation process and load balancing in the infrastructure.

Grid middleware (e.g. the Globus Toolkit (http://www.globus.org/toolkit/), Unicore [10]), especially in combination with Grid workflows, provide features similar to our approach, e.g. load balancing and distributed computing. The main difference is that we foresee a resource manager as first-class citizen in our infrastructure that handles and correlates simulation data. In Grids such a data storage is not a standard component. With OGSA [11] and the WSRF [12] Grid resources can be made available as services. Grid services provide operations to clients to steer their life time and to reserve computing time. These existing concepts had an impact on our ticket-based approach to issue or deny computing power. Nevertheless, a middleware is needed to steer work distribution on Grid services based on the processor load of resources. This task is addressed by the resource manager proposed in this paper. In fact, our general concept allows using Grid services to conduct scientific simulations and to combine Grid and Web services for simulations.

Pegasus [13] is a system to compile and run workflow-like graphs in Grids. Pegasus makes use of different catalogues that are conceptually similar to our resource manager. The site catalogue lists all participating servers; the transformation catalogue lists the software installed on these servers; and the replica catalogue contains the data items in the system. The difference to our resource manager is that Pegasus' replica catalogue does not correlate data items to a specific simulation run. The user has to know or find out himself which items are associated.

In a SOA environment, the enterprise service bus (ESB) has the task to find and bind services which are stored in a service registry [14]. ESBs do not account for the server load and for data correlation issues because business services are usually not resource-demanding and can serve many requests in parallel. The resource manager in this paper can be seen as a lightweight ESB and service registry with extensions to manage long-running, resource-demanding scientific services and the simulation data. In [15] the open source ESB Apache ServiceMix (http://servicemix.apache.org/) was extended with WSRF functionality so that resource properties can be used as service selection criteria besides the usual functional and non-functional service requirements. Simulation data storage features or fault handling patterns (e.g. service availability checks) are not implemented as foreseen in the proposed infrastructure.

In [16] the authors propose a concept for passing arbitrary data by reference in BPEL (e.g. files, database tuples). This keeps huge data sets out of the BPEL engine if not needed for navigation through the workflow (as is usually the case in scientific simulations). The data storage of the resource manager in this paper does the same but in a more light-weight fashion: Passing references is not reflected in the workflow itself, data items are *always* passed by reference. Currently, we support only data because many simulation applications rely on data stored in files. In future, we should extend the data handling towards relational databases.

3 Use Case: Simulation of Solid Bodies

The macroscopic mechanical properties of metals strongly depend on their underlying atomistic structures. Copper-alloyed α-iron, e. g., changes its material behavior during the ageing process, especially when operated at high temperatures of above 300°C. In that case, precipitates form within the iron matrix, yielding to precipitation strengthening of the material [17] followed by a decrease of the material strength as the precipitates grow further in time. In order to model this complex behavior, the growth process of precipitates, which is a diffusion based mechanism, is accounted for by a Kinetic Monte-Carlo (KMC) approach [18, 19].

A number of copper atoms is placed into a fixed body-centered cubic (bcc) iron lattice by exchanging an iron atom with a copper atom. This yields a solid solution crystal as starting configuration.

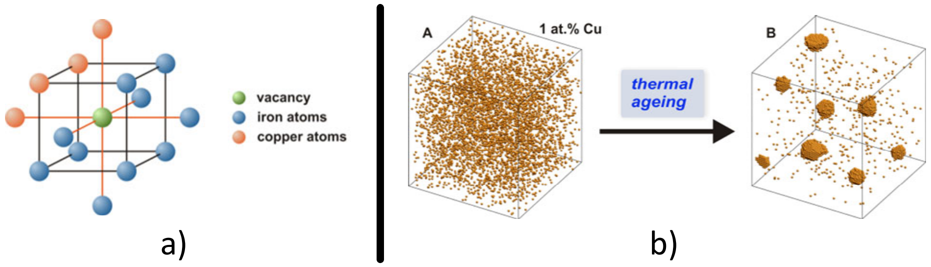


Fig. 1. Body-centered cubic crystal lattice nearest neighbors and a vacancy for one possible atom configuration (a). KMC simulation of precipitation (b). Iron atoms are transparent for better visualization. Solid solution: copper atoms are randomly distributed (A). Precipitates form during the thermal ageing process (B) [20].

The chemical interaction between atoms is described by nearest neighbor binding energies. A vacancy within the simulation box allows the movement of an atom by site exchange between the vacancy and a neighboring atom (iron, copper) (Fig. 1a). The jump activation energies depend on the local atom configuration around the vacancy. For each of the eight first neighbours of the vacancy the corresponding jump frequencies are calculated. By applying a rejection-free residence time algorithm, one of these eight possibilities is selected as the new position of the vacancy. In the long run, a series of vacancy jumps during the simulation yields the formation of precipitates with mean radii above 1 nm (Fig. 1b).

The time scale is adjusted according to the number of Monte-Carlo steps and the vacancy concentration. Periodic boundary conditions are applied in all directions in order to approximate a bulk single crystal. A more detailed description can be found in [21]. At desired time steps the atom configuration is analyzed yielding particle size distributions, mean radii of particles and the degree of supersaturation of the remaining solid solution.

3.1 Simulation Application

The simulation application opal (Ostwald-Ripening of Precipitates on an Atomic Lattice) [21] consists of five monolithic Fortran programs where the individual programs require manually created input data and are started manually. Two of them (*opalbcc* and *opalabcd*) build up the starting configuration and calculate the input values such as interaction energies, respectively. In addition to iron and copper, two more atom species can be incorporated. The precipitation process is simulated by the main application *opalmc*. After specified time intervals, opalmc generates output files containing the atom configuration (snapshot). The analysis of these atom configurations at different time steps, i.e. after specified amounts of vacancy jumps, is performed by the two programs *opalclus* and *opalxyzr* which identify the clusters within the matrix and determine size distributions and mean radii, respectively. After the analysis, the results are visualized applying commercially available software like MatLab, VMD, Rasmol, POV-Ray or Gnuplot. All in all, the overall simulation can be subdivided into four phases: preparation, simulation, analysis and visualization.

4 Service-Based Distributed Simulation Environment

Due to increasingly complex simulation models, computer simulations consume more and more computing power and storage capacity. Although the information technology improves steadily, many legacy simulation programs cannot benefit from these advancements (e.g. the opal simulation application).

Usually, computer simulation makes use of multiple tools, e.g. for calculations, visualizations, or analysis. Simulations often require manual tasks such as creation of input files, copying files between different applications that participate in a simulation, or transformation of data (e.g. merging of files). The simulation applications can conduct CPU-intensive calculations and hence exclusively engross the CPU of a machine. Using computer clusters or (public) Grids to run simulation applications would help but these infrastructures are rare and highly demanded (i.e. computing hours are difficult to obtain). In contrast, scientific institutes usually have a sufficient inventory of commodity hardware (ordinary desktop PCs). The typical work of employees (e.g. working on documents) does not use the workstations to full capacity allowing for the operation of simulation tasks in the background.

4.1 Using Services and Service Compositions to Improve This Situation

The application of SOA concepts and service compositions in these scenarios is beneficial to scientists. When scientific programs are provided as services, they can be invoked over a network using appropriate middleware. Thus, it is easy to implement distributed applications by orchestrating scientific services that are located on different resources. These resources do not have to be high-end servers—connected commodity hardware is sufficient. Furthermore, different application types can be integrated, e.g. the invocation of a visualization tool after a simulation run is finished. This contributes to the automation of manual steps, too. Thus the overall execution of an application can be sped up and can be made more efficient. Common service composition languages provide fault handling features. These can increase the robustness of simulation applications, especially if the simulation functions are implemented with a programming language that offers only restricted fault handling mechanisms. Service compositions have an additional benefit because they can also be used as (graphical) documentation of the simulation logic and help new employees and programmers to understand the overall simulation process. This is necessary due to the relatively high fluctuation of employees in scientific institutes.

4.2 Simulation Infrastructure

These above-mentioned advantages of a service-based re-engineering of legacy applications are well-known to the SOA community in theory and practice. But legacy *simulation* applications provided as services differ from services that implement business functions. The reason is that simulation applications can be long-running and processing power-demanding. Running simulation programs can easily allocate a complete computer processor. Thus, natively invoking scientific services results in the problem that busy services can get crammed with requests they cannot serve (Fig. 2a). In business environments an ESB is employed to prevent from these cases. The ESB would recognize that server A is busy because it does not respond. If the communication is asynchronous, it might be impossible for the ESB to perceive that the service is used to capacity. In both cases, the ESB *reacts* to the unavailability of a service, which means loss of time. It would be better to have a mechanism that conducts load balancing based on the processor load of servers. A resource manager that acquires services on behalf of a client can perform this task (Fig. 2b).

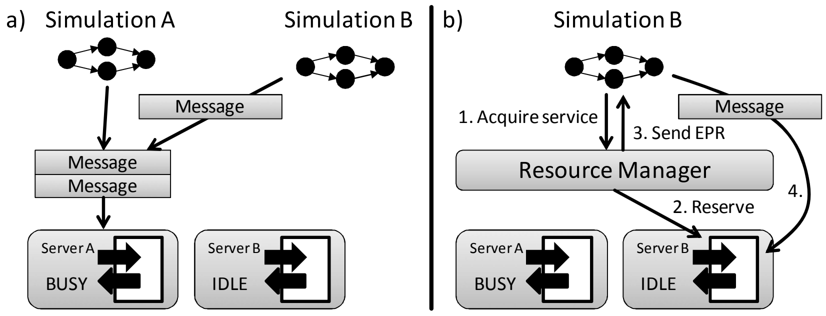


Fig. 2. Invocation of a scientific service natively (a) and with a resource manager (b)

Existing work on service compositions for scientific applications does not report on how to deal with long-running, resource-demanding services. We claim that a service-based infrastructure for scientific simulations needs a resource manager that knows the participating servers and services as well as their current load and that works as mediator between the clients and the services (Fig. 3). Note that the components can be arbitrarily distributed on participating servers and workstations. Nevertheless, having the simulation client and simulation manager installed on different engines has the advantage that the scientists can shut down their workstations without affecting the started, possibly long-running simulations. Please consider [22] for a demonstration of the prototype that implements this architecture.

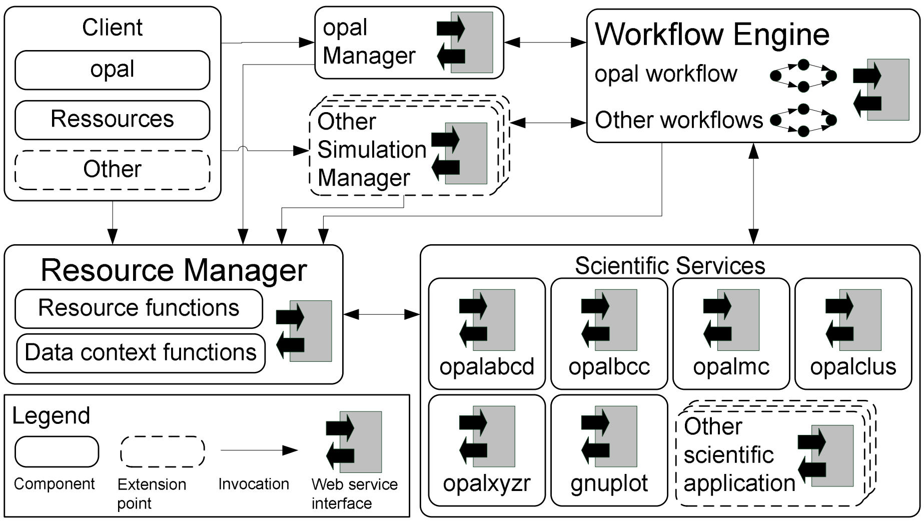


Fig. 3. Architecture of the service-based simulation infrastructure

4.3 Simulation Applications

Providing the *simulation applications* as services does not imply that existing code has to be re-implemented. However, there are cases where modifications of the simulation code are needed. This strongly depends on the granularity and the interface of the application. As a first step, it should be examined if the application consists of logical units that may be separated. This step is tedious but improves the resulting simulation application enormously [23]. The probability increases that the logical units can be reused and the simulation workflow is finer-grained. The latter enables a more detailed monitoring and more options to adjust the workflow in order to create new or adapted simulations without the need for programming and code re-compilation. Additionally, the simulation logic can be optimized with workflow analysis techniques [24]. Besides modularization of the application optimization of its interface may also be needed. It should at least be possible to invoke the application with parameters instead of hard-coding these parameters or configurations [25].

The second step is to make the application units remotely accessible as services. When using the WS technology [26], a thin WS wrapper is needed. The WS wrapper can be tailored to the application or a reusable, generic wrapper can be used that is able to invoke the application [25]. Note that the former is only possible if the simulation application is implemented in a language that allows native invocations. A WS wrapper has to be deployed on the server that offers the simulation applications. Now the simulation application can be registered with the resource manager.

With five Fortran programs and Gnuplot the simulation application opal already has an acceptable granularity (Fig. 3); there was no need to subdivide the programs into smaller units. But we extended the Fortran applications with interfaces that allow their invocation with parameters. That eliminates the need to re-compile them if the simulation configuration changes. Since there is no WS toolkit for Fortran programs, we implemented a JAVA-based WS wrapper. The approach with Java Native Access (https://jna.dev.java.net/) failed because Fortran had difficulties to process parallel write operations on files spread over several threads in a single Java process. (Curiously, this happened even for writes on different files.). The way to invoke the Fortran programs over file system commands solved these problems as each invocation creates a new Java process for execution. The re-engineering of this use case took approximately one person month for a student developer unfamiliar with Fortran 77.

4.4 Resource Manager

Besides its registry and load balancing functionality the *resource manager* (RM) also works as storage for simulation data. It provides functions to register and manage servers and scientific applications prior to or during the execution of simulations. Clients that want to use a simulation application have to ask the RM for permission. If the RM cannot find a requested service in its registry, it sends an appropriate fault message to the client. If the RM could allocate a requested service (i.e. an implementation of the service is registered and the corresponding server has computing capacities), it creates a service ticket and responds to the client with the ID of that ticket, the endpoint reference (EPR) to the service implementation, and the ticket validity in seconds. A service ticket warrants exclusive usage right to the client. The RM can recall this permission by invalidating the ticket (e.g. to enable other clients the usage of the application). This usage permission mechanism is comparable to advanced reservation techniques in Grid Computing [27]. The goal of the RM is to ensure that a server is not overloaded with requests (i.e. load balancing). If a server provides two logical processors, then it can deal in parallel with two requests for simulation applications where each allocates one processor. If the ticket validity runs out, the service is released automatically by the RM. Service clients can reset the validity time to its maximum by calling an appropriate operation of the RM. Note that the validity countdown is stopped during usage of a service. This prevents the loss of a service ticket during long-running operations. Furthermore, the RM checks the infrastructure on network partitions. A network partition between RM and a service is recognized when the service cannot acknowledge the start of an operation to the RM. The service ticket then times out; the RM removes the corresponding server and its services from the registry; and the client gets informed about ticket expiry so that it can react accordingly (e.g. by requesting the service again). A network partition might separate a service from the RM after the start of an operation was acknowledged. The RM would perceive this because we implemented a periodical availability check that would fail in this case. Again the service client is informed about the failure. The simulation applications have to implement and provide a set of operations in order to facilitate allocation, de-allocation and observation by the RM.

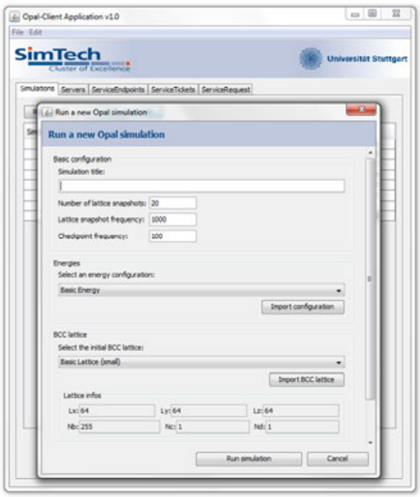
Additionally, the RM can be used as storage for simulation data (e.g. simulation parameters, configuration, results). The data may be distributed among the servers, but the RM provides a logically centralized view on them. Each simulation run gets its own simulation context where data can be saved, organized and deleted. The context correlates data items that belong to the same simulation run. This improves reproduction of simulations because the configurations and input data as well as result data are assembled and can be observed by scientists at a glance. The data items do not have to be searched for and correlated later on, which may not be possible at all since simulation applications may not have a sophisticated storage mechanism (e.g. legacy simulation applications might overwrite simulation data of former runs). Furthermore, collaboration between scientists is fostered because simulation configurations and results can easily be exchanged. Another advantage of the data storage within the simulation infrastructure is that data can be accessed and thus transmitted by reference. Only those components that are interested in a data item can load its value from the storage.

4.5 Workflow Engine, Simulation Manager and Simulation Client

The *workflow engine* executes the scientific service compositions. Due to the data RM's storage the engine is relieved of holding huge simulation data. Usually, this data is not needed for workflow navigation and it is thus sufficient to keep only the references to this data in the engine. Each simulation workflow has to be tailored to the specific simulation applications. We use Apache ODE (http://ode.apache.org/) as workflow engine and have implemented the opal workflow with BPEL. The workflow automates formerly manual tasks (e.g. starting post-processing, invocation of Gnuplot) and parallelizes the post-processing of the lattice snapshots. We have created reusable BPEL snippets for scientific service acquisition, service usage and release. This simplifies modeling workflows for other use cases or even for other BPEL-based workflow system, e.g. Sedna [6].

The *simulation managers* offer functions specific to the corresponding simulation application and workflow. With the opal manager, e.g., the client can create and manage global simulation data such as initial lattice configurations. This data is stored in the RM and can be used for several simulation runs. The opal manager can start new simulations by sending an appropriate message to the workflow engine and creates a new simulation context in the RM. All related data is stored in this context. For an asynchronous communication with the workflow engine the opal manager provides a callback were acknowledgements and other information can be sent to (e.g. the information that an error occurred in the simulation workflow). For the deletion of a simulation, the opal manager deletes the context and related data in the RM and terminates the simulation workflow instance to clean up the simulation environment.

The *simulation client* is a GUI for scientists to interact with the simulation environment. We have experienced that simulation tools are often command line-based and lack an acceptable UI. Some even have to be re-compiled if parameters change (e.g. opal, Dune). The client improves the usability of such simulation tools. The domain-specific part of the simulation client makes use of the opal manager or other simulation managers to create new simulations, provide them with input data, run and monitor the simulations, steer the simulation runs (e.g. termination, deletion), and observe intermediate and final results. The opal client allows storing input data in the RM as profile for other simulation runs (Fig. 4a). We have implemented a file explorer that shows all files related to a selected simulation run in a tree view. Additionally, we have realized a two-colored rotating 3D visualization of the lattice snapshots with the help of Java 3D (Fig. 4b). That makes it possible for scientists to observe the convergence of the simulation results during run time.



Run a new Opal simulation

Basic configuration

Simulation title:

Number of lattice snapshots: 20

Lattice snapshot frequency: 1000

Checkpoint frequency: 100

Energies

Select an energy configuration: Basic Energy

BCC lattice

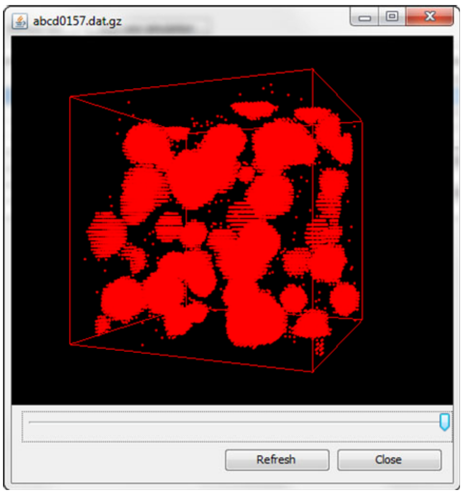
Select the initial BCC lattice: Basic Lattice (small)

Lattice infos

Lx: 64 Ly: 64 Lz: 64

Nb: 255 Nc: 1 Nd: 1

a)



b)

Fig. 4. The simulation client with the dialog to start a new opal simulation (a) and the 3D preview of the simulation snapshots (b)

Further, the RM functions of the client enable scientists to administrate the simulation infrastructure, e.g. registering servers and installed simulation applications. For each server it can be specified how many CPU cores are enabled for simulations. For each simulation service the scientist can set how many CPU cores are allocated by the service during execution. For monitoring purposes we have implemented a view that shows all service ticket requests sent to the RM and a view that lists all issued service tickets. This enables scientists to observe the status of the simulation environment and existing problems (e.g. a long list of open service requests may indicate too few available instances of a simulation service in the system).

5 Conclusions

In this paper we applied services and service compositions to re-engineer a legacy scientific application for the simulation of solid bodies. The resulting solution improves the existing application and speeds up the overall simulation by automating manual tasks and parallelizing formerly sequential work. A new GUI increases the ease-of-use for the scientists, especially with the visualization of intermediary results. The general concepts behind this use case and the extension points allow an application to other legacy simulation tools. Due to the difference between business and scientific services we had to introduce a component that steers the work distribution in the infrastructure, the resource manager. Its functionality is in parts orthogonal to that of ordinary ESBs since it accounts for the server occupation during service selection and works as storage for correlated simulation data. To the best of our knowledge none of the existing scientific workflow systems that enable the orchestration of scientific services make use of such a resource manager.

The simulation client hides the actual service compositions from the scientists. On the one hand the non-IT scientists do not get overwhelmed with technical details of the implementation. On the other hand they have to use other tools to see the workflow progress or change the workflow. In future we therefore want to integrate the simulation client in a workflow modeling and monitoring tool. Extending the resource manager so that it can install the services on demand on an idle server (with the help of provisioning techniques [27, 28]) is another open issue. Finally, a comparison of the service and workflow overhead with the automation and parallelization speedup needs to be done.

Acknowledgments. The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart. We thank Peter Binkele who contributed the MC simulation code opal to our work.

References

1. Akram, A., Meredith, D., Allan, R.: Evaluation of BPEL to Scientific Workflows. In: Proc. of the 6th IEEE International Symposium on Cluster Computing and the Grid (2006)
2. Barga, R., Jackson, J., Araujo, N., et al.: The Trident Scientific Workflow Workbench. In: Proc. of the IEEE International Conference on eScience (2008)
3. Goerlach, K., Sonntag, M., Karastoyanova, D., et al.: Conventional Workflow Technology for Scientific Simulation. In: Yang, X., Wang, L., Jie, W. (eds.) Guide to e-Science, Springer, Heidelberg (2011)
4. Sonntag, M., Karastoyanova, D., Deelman, E.: Bridging the Gap Between Business and Scientific Workflows. In: Proc. of the 6th IEEE Int. Conf. on e-Science (2010)
5. OASIS: Business Process Execution Language (BPEL) Version 2.0 (2007), `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`
6. Wassermann, B., Emmerich, W., Butchart, B., Cameron, N., Chen, L., Patel, J.: Sedna: A BPEL-Based Environment For Visual Scientific Workflow Modelling. In: Taylor (ed.) Workflows for e-Science, pp. 428–449. Springer, Heidelberg (2007)
7. Scherp, G., Höing, A., Gudenkauf, S., Hasselbring, W., Kao, O.: Using UNICORE and WS-BPEL for Scientific Workflow Execution in Grid Environments. In: Lin, H.-X., Alexander, M., Forsell, M., Knüpfer, A., Prodan, R., Sousa, L., Streit, A. (eds.) Euro-Par 2009. LNCS, vol. 6043, pp. 335–344. Springer, Heidelberg (2010)
8. Wassink, I., Ooms, M., van der Vet, P.: Designing Workflows on the Fly Using e-BioFlow. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 470–484. Springer, Heidelberg (2009)
9. MPI Forum: MPI: A Message-Passing Interface Standard Version 2.2 (2009), `http://www.mpi-forum.org/`
10. Streit, A., Bala, P., Beck-Ratzka, A., et al.: UNICORE 6 – Recent and Future Advancements. Forschungszentrum Jülich Zentralbibliothek (2010) ISSN 0944-2952
11. Foster, I., Kesselman, C., Nick, J.M., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration (2002), `http://www.globus.org/alliance/publications/papers/ogsa.pdf`
12. Foster, I., Frey, J., Graham, S., et al.: Modeling Stateful Resources with Web Services (2004), `http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf`

13. Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M.-H., Vahi, K., Livny, M.: Pegasus: Mapping Scientific Workflows onto the Grid. In: Dikaiakos, M.D. (ed.) AxGrids 2004. LNCS, vol. 3165, pp. 11–20. Springer, Heidelberg (2004)
14. Chappell, D.: Enterprise Service Bus. O'Reilly Media, Sebastopol (2004)
15. Wiese, A.: Konzeption und Implementierung von WS-Policy- und WSRF-Erweiterungen für einen Open Source Enterprise Service Bus. Diploma Thesis No. 2664, University of Stuttgart (2008)
16. Wieland, M., Goerlach, K., Schumm, D., Leymann, F.: Towards Reference Passing in Web Service and Workflow-based Applications. In: Proc. of the 13th IEEE Enterprise Distributed Object Conference (2009)
17. Kizler, P., Uhlmann, D., Schmauder, S.: Linking Nanoscale and Macroscale: Calculation of the Change in Crack Growth Resistance of Steels with Different States of Cu Precipitation Using a Modification of Stress-strain Curves Owing to Dislocation Theory. Nuclear Engineering and Design 196, 175–183 (2000)
18. Schmauder, S., Binkele, P.: Atomistic Computer Simulation of the Formation of Cu-Precipitates in Steels. Computational Materials Science 24, 42–53 (2002)
19. Soisson, F., Barbu, A., Martin, G.: Monte-Carlo Simulations of Copper Precipitates in Dilute Iron-Copper Alloys During Thermal Ageing and Under Electron Irradiation. Acta Materialia 44, 3789–3800 (1996)
20. Molnar, D., Binkele, P., Hocker, S., Schmauder, S.: Multiscale Modelling of Nano Tensile Tests for Different Cu-precipitation States in α-Fe. In: Proc. of the 5th Int. Conf. on Multiscale Materials Modelling, pp. 235–239. Fraunhofer Verlag (2010)
21. Binkele, P., Schmauder, S.: An atomistic Monte Carlo simulation for precipitation in a binary system. International Journal for Materials Research 94, 1–6 (2003)
22. Sonntag, M.: Workflow-based Simulation of Solid Bodies. Prototype demo (2010), `http://www.iaas.uni-stuttgart.de/forschung/projects/simtech/downloadsE.php#opalVideo`
23. Rutschmann, J.: Generisches Web Service Interface um Simulationsanwendungen in BPEL-Prozesse einzubinden. Diploma Thesis No. 2895, University of Stuttgart (2009)
24. Leymann, F., Roller, D.: Production Workflow – Concepts and Techniques. Prentice Hall, Englewood Cliffs (2000)
25. Hotta, S.: Ausführung von Festkörpersimulationen auf Basis der Workflow Technologie. Diploma Thesis No. 3029, University of Stuttgart (2010)
26. Weerawarana, S., Curbera, F., Leymann, F., et al.: Web services platform architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and more. Prentice Hall, Englewood Cliffs (2005)
27. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a New Computing Infrastructure, 2nd edn. Morgan Kaufmann, San Francisco (2004)
28. Mietzner, R., Leymann, F.: Towards Provisioning the Cloud: On the Usage of Multi-Granularity Flows and Services to Realize a Unified Provisioning Infrastructure for SaaS Applications. In: Proc. of the Int. Congress on Services (2008)