Original published: M. Hofmann, F. Ospald, H. Schmidt, and R. Springer. Programming support for the flexible coupling of distributed software components for scientific simulations. In A. Holzinger, T. Libourel, L. A. Maciaszek, and S. J. Mellor, editors, *Proceedings of the 9th International Conference on Software Engineering and Applications (ICSOFT-EA 2014)*, pages 506–511. SciTePress, 2014. Online available at `http://dx.doi.org/10.5220/0005105405060511`.

Programming Support for the Flexible Coupling of Distributed Software Components for Scientific Simulations

Michael Hofmann[1], Felix Ospald[2], Hansjörg Schmidt[2], and Rolf Springer[2]

[1] *Department of Computer Science, Chemnitz University of Technology, Chemnitz, Germany*

[2] *Department of Mathematics, Chemnitz University of Technology, Chemnitz, Germany*

Keywords: distributed simulations, data coupling, parallel computing

Abstract: In this article, we investigate the flexible coupling of distributed software components that are required for an optimization process of lightweight structures built from hybrid materials. The software components include computationally intensive applications for the simulation of hybrid structures, control applications for implementing the optimization process as well as data-oriented applications for the generation, management, and visualization of simulation data. The participating software components and application programs are described to demonstrate their strongly varying functionalities as well as the diversity of data exchange methods that need to be considered for the data coupling. Furthermore, we present the design and usage of a software library with transparent data coupling mechanisms for software components that are flexibly distributed among different computing resources.

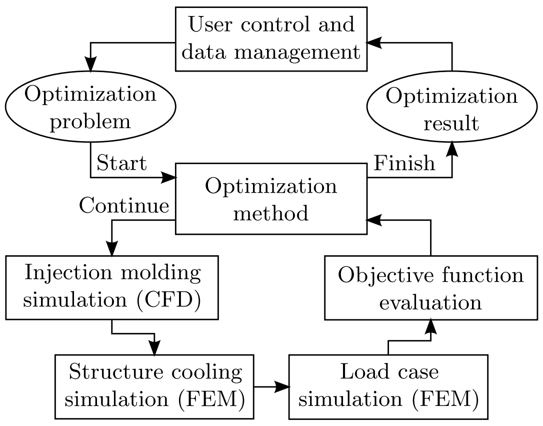
1 Introduction

The coupling of scientific simulations is an increasingly popular method for the simulation of complex dynamic physical problems. Especially, strongly inter-disciplinary areas, such as climate research or mechanical engineering, often lead to simulation models that are independently developed, but need to be integrated in larger simulation environments later. Technical approaches for the required model coupling include dedicated toolkits (Larson et al., 2005), domain-specific languages (Bulatewicz, 2006; Kohn et al., 2000), and generalized interfaces (Gregersen et al., 2007). By focusing on the data exchange, model-independent software components, such as for the generation, management, or visualization of data can also be integrated. However, the variety of applications and their flexible execution in a distributed environment pose a challenge to the data coupling.

In this article, we investigate the requirements towards an efficient programming support for the data coupling of scientific simulations that are executed in a distributed computing environment. As an application example, the optimized design of lightweight structures within the research project MERGE[1] is considered. The optimization process developed includes both the simulation of the manufacturing of hybrid structures and the calculation of their operating load cases. These computationally intensive applications are executed several times during the optimization process, thus requiring the usage of high-performance computing platforms. Data coupling is required between the simulation applications as well as with additional applications that implement, for example, the optimization method, an interactive user control and a central data repository. Depending on the optimization problem as well as on the required computing resources, a flexible execution and distribution of the applications is required, thus leading to different programming efforts for their data coupling. For example, the applications have to be loosely coupled (e. g., through network communications) when executed in a distributed computing environment or tightly coupled (e. g., through direct function calls) when executed on a single compute node. We present the design of a software library that supports a transparent data coupling of such flexible distributed applications and show a usage example to demonstrate our approach.

The rest of this article is organized as follows. Section 2 describes the distributed software components for the optimization of hybrid structures. Section 3 presents the design and usage of the software library. Section 4 discusses related work and Sect. 5 concludes the article.

[1]MERGE Technologies for Multifunctional Lightweight Structures, `http://www.tu-chemnitz.de/merge`


User control and data management
Optimization problem
Optimization result
Start
Finish
Optimization method
Continue
Injection molding simulation (CFD)
Objective function evaluation
Structure cooling simulation (FEM)
Load case simulation (FEM)
Figure 1: Overview of the optimization process.

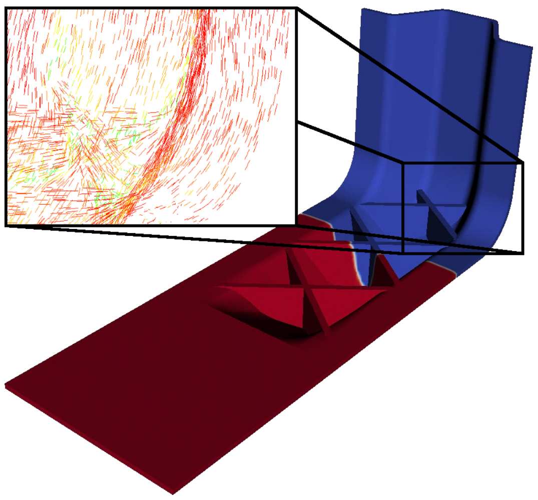


Figure 2: Illustration of the injection molding simulation with resulting fiber orientation details.

2 Distributed Software Components for Optimizing Hybrid Structures

The distributed simulation and optimization of hybrid structures is used as an application example that requires a flexible coupling of a variety of software components. In the following, the optimization process and its software components are described.

2.1 Optimization of Hybrid Structures

The goal of the optimization process is the optimization of manufacturing parameters for hybrid structures. Figure 1 gives an overview of the optimization process. Configuring and executing optimization problems will be performing through a user interface, which will also be used to manage the input and output data of optimization runs. Each optimization run starts with an optimization problem that specifies the geometry of the structure, the objective function and constraints for the optimization, and parameters, for example, about the materials, the manufacturing process, and the operating load cases. The optimization method is implemented within a separate software component that executes an optimization loop. In each iteration, the optimization method selects specific values for the parameters to be optimized, starts the simulations, and evaluates the objective function to decide whether the optimization is finished or not. The manufacturing by injection molding is simulated with a computational fluid dynamics (CFD) application (see Sect. 2.2). The cooling of the structure and the operating load cases are simulated with a finite element method (FEM) application (see Sect. 2.3).

Figure 2 shows an injection molding simulation where the material is injected into the lower left face and the structure is about half filled. The test case represents an existing injection molding tool and will be used to compare the simulation and optimization results with manufactured parts. Manufacturing parameters to be optimized include the position, diameter, angle, and fill rate for injecting the material.

2.2 CFD Simulation of the Manufacturing Process

The hybrid structures will be manufactured by injection molding, which represents one of the most economically important processes for the mass-production of plastic products. The parts are produced by injecting molten plastic into a mold, followed by a cooling process. Fillers, such as glass and carbon fibers are mixed in to improve the mechanical properties of the parts. The injection molding process is simulated with a customized open-source CFD application based on OpenFOAM, a C++ library implementing the finite volume method (Jasak et al., 2007) parallelized with MPI. However, our general optimization process will also be capable of employing alternative simulations, for example, with specialized closed-source applications (Niedziela et al., 2013).

Our OpenFOAM solver features a compressible two phase flow with heat transfer and fiber orientation calculation. The change of the fiber distribution caused by the flow is modelled by the Folgar-Tucker Equation (Tucker and Advani, 1994). Input data of the simulation are the structure geometry, the material properties, and the parameters of the injection molding process. The simulation is complete when the mold is filled. Simulation results are the fiber orientation field and the temperature field, which are the input data for the FEM simulation (see Sect. 2.3).

2.3 FEM Simulation of Cooling and Load Cases

An FEM application simulates the cooling of the structure and the operating load cases. The cooling leads to residual stresses within the material and to a shrinking of the structure. The load case simulation results in displacements and stresses of the structure. Both simulations are performed with an in-house adaptive 3D FEM application (Beuchler et al., 2001) parallelized with OpenMP. However, our general optimization process will also be capable of employing alternative simulations with commercial closed-source FEM applications, such as ANSYS[2] or Abaqus[3].

The adaptive FEM starts with a coarse mesh, which is adaptively refined based on residual type error indicators (Verfürth, 1999). This approach leads to high precision results, but with reduced computational costs. Input data of the simulation are the structure geometry, the material properties, and the load case to simulate. Additionally, the fiber orientation and the temperature distribution within the structure calculated by the CFD simulation are used. Simulation results are the displacement field and the resulting stresses, which will be used for evaluating the objective function of the optimization process.

2.4 Optimization Objectives

An iteration of the optimization loop can be very time consuming. Furthermore, the results of the numerical simulations are often discontinuous and noisy with respect to the optimization parameters and derivatives are usually not available. The optimization may have multiple objectives regarding the manufacturing process (e. g., minimization of production costs) and structure properties (e. g., maximization of the strength, stiffness, or durability). Thus, in general, we will have to employ high-dimensional, derivative-free, multi-objective optimization algorithms to determine Pareto optimal solutions. These algorithms will either be implemented separately in Python or employed through existing software frameworks, such as OpenMDAO (Heath and Gray, 2012).

In each iteration of the optimization loop, manufacturing and load case simulations have to be performed for several different settings of the parameters to be optimized. The simulation runs for these parameter settings are independent from each other and can therefore be distributed, for example, on various computing resources. The number of independent simulation runs depends strongly on the number of optimization parameters and on the optimization method.

[2]http://www.ansys.com/

[3]http://www.simulia.com/

2.5 User Control and Data Management

In addition to the simulation applications, several software components will be required to handle the data involved in the optimization process. Mesh generators, such as Gmsh (Geuzaine and Remacle, 2009), are required to model the structure geometries and to generate the meshes for the simulations. Data repositories store the simulation data and provide a project-oriented storage for simulation and optimization parameters. Applications, such as ParaView (Henderson Squillacote, 2008) or VTK (Schroeder et al., 2006), are used for the visualization of simulation results. The usage of the optimization process will be supported by a web interface, that allows an user-friendly configuration and execution of optimization runs.

3 Simulation Component and Data Coupling (SCDC) Library

The implementation of the optimization process will be support by a communication library that eases the coupling of different software components. In the following, the communication library, its programming interface and an usage example are presented.

3.1 Requirements and Solutions

The various applications involved in the optimization process described in Sect. 2 lead to different requirements towards a suitable programming support:

Variety of software components: The considered software components differ strongly with regard to their functionalities as well as their specific implementations as application programs. To unify the general coupling of the software components, the SCDC library enforces the following usage model: Each software component can act as an *SCDC service* that provides access to *datasets*. Each software component can act as an *SCDC client* to access the datasets of other software components acting as services. An URI-based addressing scheme is used to identify single services and datasets. The datasets of a service are made available by *data providers*, which define the specific content and functionality of a dataset. This includes data providers with predefined purposes (e. g., for accessing the local file system or a MySQL database) as well as generic data providers that are configured with hook functions. Each service can contain several data providers, which are distinguished through individual base paths within the URI-based addressing scheme.

Flexible execution on distributed resources: The execution of the software components should be flexibly distributed on various computing resources. This means that the software components of the optimization process should be executed, for example, either entirely on a single compute node or separately on dedicated computing resources that fit to the different applications (e. g., storage server, HPC cluster, web server). The SCDC library omits the need for expensive adaptations of each software component to a specific execution scenarios. A service uses the same library functions to provide access to its dataset through different data coupling mechanisms (e. g., direct memory buffer access, inter-process or network communication). Similarly, the access from a client to a service uses the same library functions, independent from the data coupling mechanism. Accessing different services requires only to change the URI address that identifies the target service. All adaptations necessary for exploiting different data coupling mechanisms are encapsulated in the library.

Diversity of data access methods: The different software components and their alternative realizations (e. g., different data storage components or alternative simulation applications) lead to a heterogeneous landscape of application programs with a large diversity of supported data access methods. The SCDC library provides a C and Python interface for the integration into application programs which allow modifications of their source code. With these interfaces, the data access can be performed directly through memory buffers provided to the library functions while at the same time the underlying data exchange is mapped, for example, to network communication. Additionally, the Python interface is used for wrapping closed-source (e. g., commercial) application programs that employ usually a file-based data access. To provide such an application as a service, a dedicated Python program will be used for setting up the service with the SCDC library, executing the specific application program, and managing its input and output files.

3.2 Programming Interface and Usage

The programming interface of the SCDC library consists of service and client functions:

Service functions: A software component that acts as a service has to specify the data coupling mechanisms available for client accesses. Direct coupling is enabled as default and connects all commands executed by a client to direct function calls within a service. Further connection-oriented coupling (e. g., with communication based on Unix Domain sockets or TCP sockets) can be enabled with the `scdc_nodeport_open/close/start/stop` functions. Each data coupling mechanism has to be opened and closed separately and can be started and stopped temporarily. The `scdc_dataprov_open/close` functions are used to add data providers to a service.

Client functions: A software component that acts as a client uses the `scdc_dataset_open/close/cmd` functions to access the datasets provided by a service. The following URI-based addressing schemes are currently used to identify a dataset to be opened:

- `scdc:///<path>` to access datasets of the same software component through direct coupling.
- `scdc+uds://<socketname>/<path>` to access datasets of software components executed on the same compute node through inter-process communication with Unix Domain sockets. The socket to be used is specified with `<socketname>`.
- `scdc+tcp://<hostname>/<path>` to access datasets of software components executed on different compute nodes through network communication with TCP sockets. The compute node of the service is specified with `<hostname>`.

The specification of `<path>` for identifying a specific dataset is the same for all three schemes. A first part of `<path>` usually represents the base path that selects a specific data provider of a service and the remaining seconds part of `<path>` identifies a specific dataset of this data provider. However, the specific kind of data represented by a dataset depends on the data provider. For example, a dataset represents a file system directory for the data provider accessing the local file system while it represents a database table for the data provider accessing a MySQL database.

Usage example: Figure 3 shows an usage example for the SCDC library with two software components acting as a storage service A (top) and as a client with integrated storage service B (bottom). The storage service A is executed on a compute node `hostA` and stores the data in its local file system. In lines 2–3, a data provider of type "`fs`" is created that can be accessed with the base path "`store`" and uses the specified storage directory. In line 5, access to the

```
 1 /* storage service A on hostA */
 2 dp = dataprov_open("store", "fs",
 3   "path_to_storage_directory");
 4
 5 np = nodeport_open("tcp");
 6 nodeport_start(np, 0);
 7
 8 /* keep the service running */
 9 while (getchar() != 'q');
10
11 nodeport_stop(np);
12 nodeport_close(np);
13
14 dataprov_close(dp);
```

```
14 /* client & db storage service B */
15 dp = dataprov_open("store", "mysql",
16   "dbserver:dbuser:dbpasswd:dbname");
17
18 dataset_input_create(&in, "buffer",
19   buf, buf_size);
20
21 if (dbstore) uri = "scdc:///store";
22 else uri = "scdc+tcp://hostA/store";
23
24 /* open dataset & store data 'XY' */
25 ds = dataset_open(uri);
26 dataset_cmd(ds, "put XY", &in, NULL);
27 dataset_close(ds);
28
29 dataset_input_destroy(&in);
30
31 dataprov_close(dp);
```

Figure 3: Usage example for the SCDC library with two software components, i. e. storage service A (top) and client with integrated storage service B (bottom). The prefix `scdc_` of library functions is omitted due to space reasons.

storage service is opened through connection-oriented coupling with TCP sockets. The corresponding TCP server is started in a non-blocking way (line 6). Thus, it is necessary to keep the main thread of the service running and to provide an appropriate mechanism for terminating the service (line 9). Finally, the TCP server and the data provider are stopped (lines 11–14).

The client with integrated storage service B shown in Fig. 3 (bottom) stores the data within a MySQL database. Thus, in lines 15–16, a data provider of type "`mysql`" is created by specifying the required database access credentials. The client stores data from a memory buffer either with storage service A or B. In lines 18–19, an input data object `in` is prepared such that it reads from the memory buffer `buf`.

The selection between storage service A and B is achieved by a differing URI address for the dataset of the storage operation (lines 21–22). The storage operation itself is performed by opening the corresponding dataset (line 25) and performing a `put` command to store the data from the input data object (line 26). Since there is no output expected for the command, the usage of an output data object is omitted. The handle `ds` of the opened dataset might be used to perform several commands before it is closed (line 27). Finally, the input data object is destroyed and the data provider representing the MySQL database is stopped. This usage example demonstrates how a software component can access various alternative software components without adaptations to its implementation. Furthermore, the different data coupling mechanisms (i. e., direct memory buffer access or network communication with TCP sockets) are transparently hidden within the SCDC library.

4 Related work

Environmental research is one of the most prominent areas for the coupling of simulation models, as it involves a variety of models from disciplines, such as atmospheric sciences, hydrology, geology, chemistry, and ecology. The resulting need for interoperable software components has led to a large number of architectures, frameworks, and toolkits specifically designed to support the coupling of simulation models. Established approaches for high performance computing are, for example, the Earth System Modelling Framework (Hill et al., 2004), the Common Component Architecture (Bernholdt et al., 2006), and the Model Coupling Toolkit (Larson et al., 2005). However, a proper characterization of the approaches and their roles for model coupling is very hard due to their broad functionalities, various implementations, and supportive tools. A more detailed overview of the approaches and their features is given by Jagers (Jagers, 2010) and Dunlap et al. (Dunlap et al., 2013).

Coupling via an I/O infrastructure represents a low-level alternative to model coupling. Specialized communication libraries, such as PSMILe from the OASIS framework (Redler et al., 2010) or the parallel coupler PALM (Piacentini et al., 2011), provide functions for the data exchange between parallel software components. Since these functions are mapped to MPI operations, the data exchange is restricted to MPI programs in a uniform HPC software environment. The Typed Data Transfer (TDT) library (Linstead, 2004) circumvents this limitation by using files and sockets as alternative data exchange methods next to MPI. The SCDC library provides a similar encapsulation of different data exchange methods. However, the focus on flexible distributed software components requires to support intra-process, inter-process, and network communication. Furthermore, the existing approaches usually provide two-sided communication operations that need to be actively invoked by each software component. Instead, the SCDC library provides an infrastructure for implementing clients and services where commands, such as put and get, represent remote data accesses.

5 Conclusion

We have described the distributed software components required for the simulation and optimization of lightweight structures. Especially the data coupling between HPC simulation codes and data management applications has to be improved to achieve an automated optimization process. The software library proposed will support the development by replacing the common file-based data exchange of these applications. By encapsulating different data exchange mechanisms into the library, a flexible distribution of the software components among different computing resources will be achieved without requiring expensive adaptations to the application codes.

Acknowledgment

This work was performed within the Federal Cluster of Excellence EXC 1075 “MERGE Technologies for Multifunctional Lightweight Structures” and supported by the German Research Foundation (DFG). Financial support is gratefully acknowledged.

REFERENCES

Bernholdt, D., Allan, B., Armstrong, R., Bertrand, F., Chiu, K., Dahlgren, T., Damevski, K., Elwasif, W., Epperly, T., Govindaraju, M., Katz, D., Kohl, J., Krishnan, M., Kumfert, G., Larson, J., Lefantzi, S., Lewis, M., Malony, A., Mclnnes, L., Nieplocha, J., Norris, B., Parker, S., Ray, J., Shende, S., Windus, T., and Zhou, S. (2006). A component architecture for high-performance scientific computing. *Int. J. High Performance Computing Applications*, 20(2):163–202.

Beuchler, S., Meyer, A., and Pester, M. (2001). SPC-PM3AdH v1.0 - Programmer’s manual. *Preprint SFB/393 01-08, TU-Chemnitz.*

Bulatewicz, T. (2006). A domain-specific language for model coupling. In *Proc. of the Winter Simulation Conf.*, pages 1091–1100. IEEE.

Dunlap, R., Rugaber, S., and Mark, L. (2013). A feature model of coupling technologies for earth system models. *Computers & Geosciences*, 53:13–20.

Geuzaine, C. and Remacle, J.-F. (2009). Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *Int. J. Numerical Methods in Engineering*, 79(11):1309–1331.

Gregersen, J., Gijsbers, P., and Westen, S. (2007). OpenMI: Open modelling interface. *J. Hydroinformatics*, 9(3):175–191.

Heath, C. and Gray, J. (2012). OpenMDAO: Framework for flexible multidisciplinary design, analysis and optimization methods. In *Proc. of the 8th AIAA Multidisciplinary Design Optimization Specialist Conf.*

Henderson Squillacote, A. (2008). *The ParaView guide: A parallel visualization application*. Kitware.

Hill, C., DeLuca, C., Balaji, V., Suarez, M., and da Silva, A. (2004). The architecture of the earth system modeling framework. *Computing in Science & Engineering*, 6(1):18–28.

Jagers, H. (2010). Linking data, models and tools: An overview. In *Proc. of the Int. Congress on Environmental Modelling and Software (iEMSs’10).*

Jasak, H., Jemcov, A., and Tukovic, Z. (2007). OpenFOAM: A C++ library for complex physics simulations. In *Proc. of the Int. Workshop on Coupled Methods in Numerical Dynamics (CMND’07).*

Kohn, S., Kumfert, G., Painter, J., and Ribbens, C. (2000). Divorcing language dependencies from a scientific software library. In *Proc. of the 10th SIAM Conf. on Parallel Processing for Scientific Computing*. SIAM.

Larson, J., Jacob, R., and Ong, E. (2005). The Model Coupling Toolkit: A new Fortran90 toolkit for building multiphysics parallel coupled models. *Int. J. High Performance Computing Applications*, 19(3):277–292.

Linstead, C. (2004). Typed Data Transfer (TDT) user’s guide.

Niedziela, D., Tröltzsch, J., Latz, A., and Kroll, L. (2013). On the numerical simulation of injection molding processes with integrated textile fiber reinforcements. *J. Thermoplastic Composite Materials*, 26(1):74–90.

Piacentini, A., Morel, T., Thévenin, A., and Duchaine, F. (2011). O-PALM: An open source dynamic parallel coupler. In *Proc. of the IV Int. Conf. on Computational Methods for Coupled Problems in Science and Engineering*.

Redler, R., Valcke, S., and Ritzdorf, H. (2010). OASIS4 – A coupling software for next generation earth system modelling. *Geoscientific Model Development*, 3(1):87–104.

Schroeder, W., Martin, K., and Lorensen, B. (2006). *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*. Kitware.

Tucker, C. and Advani, S. (1994). Processing of short-fiber systems. *Flow and Rheology in Polymer Composites Manufacturing*, pages 147–147.

Verfürth, R. (1999). A review of a posteriori error estimation techniques for elasticity problems. *Computer Methods in Applied Mechanics and Engineering*, 176(1–4):419–440.