

## INTEGRATING GENERIC FEM SIMULATIONS INTO COMPLEX SIMULATION APPLICATIONS

ROBERT DIETZE<sup>†</sup>, MICHAEL HOFMANN<sup>†</sup>, AND GUDULA RÜNGER<sup>†</sup>

**Abstract.** In this article, the efforts for integrating alternative FEM codes into a complex simulation application from the area of engineering optimisation are described. The application area, its participating software components, and their interactions are presented. The integration of two different FEM codes is demonstrated based on a dedicated FEM data conversion component. Performance results are shown to investigate the overhead caused by the data conversion.

**Key words.** FEM simulations, data conversion, component-based development, distributed simulations

**1. Introduction.** The development of scientific applications for complex problems often leads to complicated program codes that are hard to maintain and less portable in terms of their performance on different hardware platforms. Component-based development approaches can provide means to handle this complexity, especially if several independently developed application programs have to be integrated into a single complex simulation [12]. Important aspects of these approaches are the implementation of new components or the integration of existing components, the efficient data exchange between components based on various communication methods, and the flexibly distributed execution of components on different hardware platforms. A further major challenge is to ensure the interchangeability of specific components, for example, to perform compute-intensive tasks, such as finite element method (FEM) simulations, with different application programs.

Having the ability to choose flexibly between different application programs provides several advantages. Program codes with different application functionalities, for example, might provide alternative solution methods that lead to different result precision or support specific simulation conditions. If the program codes provide the same application functionality, then it might be preferable to use codes that reduce costs, for example, in terms of execution time, hardware utilisation, or software licenses. The flexible integration of FEM codes into a component-based scientific simulation needs support by a generic FEM simulation component that can invoke different FEM codes. This approach would allow users of complex simulation applications, e.g. domain specialists such as mechanical engineers, to focus on the design of their simulation problems and solution methods. The usage and integration of specific program codes as well as their efficient execution on dedicated platforms, such as high performance computing clusters, is then left to the application developers.

The complex simulation which we consider is an application from mechanical engineering for optimising lightweight structures based on numerical simulations. This class of applications is extensively studied in the research project MERGE<sup>1</sup>. The goal is to perform simulation-based optimisations of the design and the manufacturing of fibre-reinforced plastics [10]. The overall complex simulation application consists of various program components, such as computationally intensive numerical simulations, control programs for implementing the optimisation process as well as data-oriented programs for the generation, management, and visualisation of the sim-

---

<sup>†</sup>Department of Computer Science, Technische Universität Chemnitz, Chemnitz, Germany

<sup>1</sup>MERGE Technologies for Multifunctional Lightweight Structures, <http://www.tu-chemnitz.de/merge>

ulation data. FEM codes are used to simulate the cooling of the manufactured parts, which leads to residual stresses and deformations, and for the characterisation of their mechanical properties with specific operating load cases. To achieve an interchangeability of the FEM simulation component, it is required to implement data conversions for the FEM input data formats and to utilise different FEM codes and their specific execution platforms. The FEM input data formats are usually text-based and comprise information, such as the geometry and the material properties of the part to be simulated and the boundary conditions of the desired solution. The FEM codes range from open source codes to closed commercial or proprietary codes while as execution platforms usually both Linux/Unix-based or Windows-based systems are used.

In this article, we present the integration of a generic FEM simulation component into a component-based simulation application for the optimisation of lightweight structures. The main contributions are as follows: We describe the major components of the complex simulation application as well as the interactions between these components based on a service-oriented approach. The integration of generic FEM simulations is mainly achieved with a dedicated FEM data conversion component. We present the design of this conversion component and describe its support for two different FEM input data formats. Finally, we describe the implementation of the program components of the simulation-based optimisation application and their interactions based on the Simulation Component and Data Coupling (SCDC) library [12]. The SCDC library supports different data exchange methods, such as direct function calls, inter-process communication, and network communication. Thus, using the SCDC library allows for a flexibly distributed execution of the program components among different hardware platforms without requiring additional programming efforts.

The rest of this article is organised as follows. Section 2 gives an overview of the complex simulation application for optimising lightweight structures. Section 3 presents the data conversion between different input formats of FEM codes. Section 4 describes the component-based approach for implementing the distributed execution of the program components of the complex simulation. Section 5 presents results for estimating the overhead of the FEM data conversion in comparison to the execution of the FEM codes. Section 6 discusses related work and Section 7 concludes the article.

**2. A complex simulation application for the optimisation of lightweight structures.** The optimisation of lightweight structures to be developed in the project MERGE is performed with numerical simulations for manufacturing and using fibre-reinforced plastics. The simulation-based approach and the component-based implementation of the resulting complex simulation application is described in the following.

**2.1. Optimising lightweight structures.** The lightweight structures considered in the project MERGE are plastic parts that are manufactured by injection moulding. Fillers, such as short glass or carbon fibres, are mixed into the plastic to improve the mechanical properties of the parts. A computational fluid dynamics (CFD) simulation is used to simulate the manufacturing process of injecting the molten plastic and the fibres into a mould. The density and orientation of the fibres within the manufactured parts have a strong influence on their mechanical properties. The results of the CFD simulation characterise the fibre distribution within the parts and are used to model the material properties of such short fibre-reinforced plastics.

The CFD simulation is finished when the mould is filled with the molten material, thus leading also to a temperature distribution within the manufactured part. A thermal analysis simulation uses the temperature distribution as a starting point to simulate the cooling to room temperature. This cooling process can lead to residual

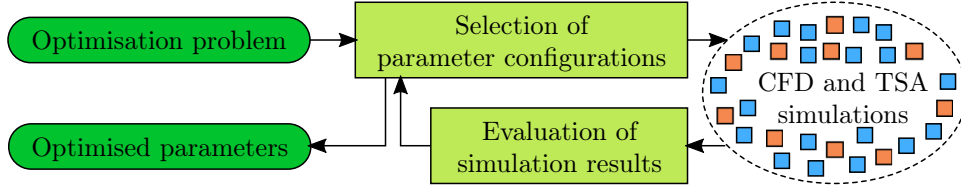


FIGURE 2.1. Overview of the coarse structure of the optimisation process for designing lightweight structures.

stresses in the solid material and to deformations of the manufactured part. The resulting information about the geometrical and material properties of the manufactured part are finally used to simulate its behaviour in different operating load cases using structural analysis simulations. Both thermal and structural analysis (TSA) simulations are performed with finite element method (FEM) programs.

The overall goal of the complex simulation application is to optimise the properties of the lightweight structures. Figure 2.1 shows a coarse overview of this optimisation process. Based on an optimisation problem defined by the user of the complex simulation application, an optimisation method selects specific values for the parameters to be optimised. For short-fibre reinforced plastics, material and manufacturing parameters, such as the fibre percentage or the injection position, are varied. Each selected parameter configuration is then used to simulate the manufacturing, the cooling, and the usage of the corresponding plastic part. In this step, usually about 10–100 simulation tasks have to be computed with dedicated CFD and TSA simulation programs. However, the specific number of tasks depends on the number of parameters to optimise, the utilised optimisation method, and the load cases to consider. The simulation results with the different parameter configurations are then evaluated, such that the optimisation method can either select further parameter configurations to simulate or finish with the optimised parameters. A Kriging metamodel approach is used for the global optimisation [13]. A more detailed overview of the simulation and optimisation approaches developed in the research project MERGE is given in [8].

In this work, we concentrate on the utilisation of FEM codes for the simulation of operating load cases. This can be used, for example, to perform parameter studies with successively changing loads or to compare different solution methods supported by FEM programs. As a primary example of an FEM program, we use an in-house adaptive FEM code called SPC-FEM [5] which has been further developed according to the needs of the project MERGE. The input data for the FEM simulations to perform is generated in the custom data format of the SPC-FEM code. Additional commercial FEM codes, such as ANSYS<sup>2</sup>, should be employed as alternative or complementary FEM methods. Thus, it is required to integrate appropriate data conversions and program executions into the complex simulation application.

**2.2. Component-based application with generic FEM integration.** The optimisation of lightweight structures described in the previous section involves a variety of different application programs. To achieve a sustainable solution that allows a continuous adaption to new usage scenarios and a flexible utilisation of distributed execution platforms, the overall complex simulation application is separated into individual software components as follows:

<sup>2</sup>[www.ansys.com](http://www.ansys.com)

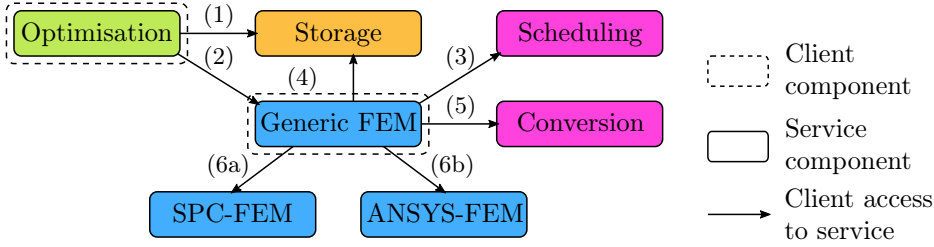


FIGURE 2.2. Overview of the components for the service-oriented implementation of the simulation application for lightweight structures. The interactions (1)–(6) represent client accesses to services.

- The **optimisation** component contains the modelling of the optimisation problem with its parameters as well as the optimisation method to be used. This component acts as a driver for the simulation application and generates the input data for the FEM simulation tasks to be performed. The execution is usually not computationally intensive and might be based on user interactions. The optimisation component is thus executed on a desktop platform.
- **Simulation** components execute the FEM simulation tasks. Separate FEM components are available to perform the simulations either with the SPC-FEM or the ANSYS-FEM code. For the integration of a generic FEM simulation, an additional generic FEM component that invokes the application-specific FEM components is provided. Since FEM simulations can be computationally intensive, it might be advantageous to execute them on HPC platforms. However, the generic FEM component might also be executed in close cooperation with the optimisation component on a desktop platform.
- Domain-specific problems that are not directly related to the simulation-based optimisation of lightweight structures are solved by auxiliary components. This includes a dedicated **scheduling** component that determines schedules for the efficient utilisation of HPC platforms. In [8], we have presented several scheduling methods for assigning simulation tasks to compute resources of a heterogeneous compute clusters such that the total time for executing all simulation tasks is minimised.
- The data conversion between different FEM input data formats is performed by a dedicated **conversion** component. The input data formats used by the SPC-FEM and ANSYS-FEM codes and the implementation of the data conversion is described in Section 3. Both the scheduling and the conversion component are mainly used by the generic FEM simulation component and, thus, are executed close to their execution platform.
- A **storage** component provides separate locations for storing the input and output data of each FEM simulation. This component remains passive and does not perform any computations. However, to support high numbers of FEM simulations with large amounts of simulation data, the storage component might be executed on a dedicated server with high storage capacities.

A service-oriented approach is used to model the interactions between the different components. Within this model, each component can be both a client that accesses other service components and a service that is accessed by other client components. Figure 2.2 illustrates the service-oriented implementation of the simulation application for the optimisation of lightweight structures. The optimisation component is invoked

by the user of the simulation application and generates the FEM simulations to be performed. Thus, this component represents a client that is activated first and then accesses other components. The execution of several FEM simulations (e.g., with different parameters, see Section 2.1) is then performed as follows:

- The input data of the FEM simulations is first transferred from the optimisation component to the storage service (1) and then the FEM simulation tasks are submitted to the generic FEM service (2).
- The generic FEM service uses information about the FEM simulation tasks and the available compute resources to set up a scheduling problem. Solving this problem is performed by a dedicated scheduling service that is accessed by the FEM service (3).
- The generic FEM service retrieves the input data of each FEM simulation task from the storage service (4) and performs the required data conversions by accessing the conversion service (5).
- The generic FEM service submits the FEM simulation tasks with its converted input data and the scheduling information about the compute resources to be used to either the SPC-FEM (6a) or the ANSYS-FEM (6b) service. The utilised service then executes its specific FEM code on the given compute resources. The FEM simulation results are gathered by the generic FEM service and then transferred to the storage service. After the generic FEM service finished all FEM simulation tasks, the submission of the FEM simulation tasks performed by the optimisation component (2) is completed.

In [12], we have presented the SCDC library that is specifically designed for implementing complex simulation applications with a component-based service-oriented approach as described in this subsection. The usage of the SCDC library for implementing the simulation application for the optimisation of lightweight structures including the flexible use of FEM codes introduced in this work is given in Section 4.

**3. Data conversions for generic FEM simulations.** The integration of generic FEM simulations into the optimisation application for lightweight structures is based on a data conversion between different FEM input data formats. In the following, a generic input data of FEM simulations as well as two specific data formats of FEM codes are described. Furthermore, the conversion between the two specific formats and their integration in a flexible conversion tool is presented.

**3.1. Generic FEM input data.** The FEM input data consists of three main parts: the geometry of the structure to be simulated, its material properties, and the boundary conditions of the desired solution. The geometry part describes the shape of the structure to be simulated and is assumed to be given as a mesh. The smallest units of a mesh are nodes given by their coordinates in a three dimensional space. Pairs of nodes can be connected to create edges. Multiple edges together form faces, e.g. four edges form a quadrilateral face. The composition of multiple faces leads to elements. Finally, the whole structure is represented by a set of elements.

The material part of the FEM input data specifies the properties of the materials the structure to be simulated consists of. Each material is described by a set of parameters of a corresponding material model. Currently, a linear elastic model is supported and the elements of the structure can use different materials (i.e., with different sets of parameter). Simulating the behaviour of a structure requires to determine the solution of an equation system that describes the state of the structure. The boundary conditions usually define constraints that need to be fulfilled by the desired solution. For example, in the FEM input data, the boundary conditions can

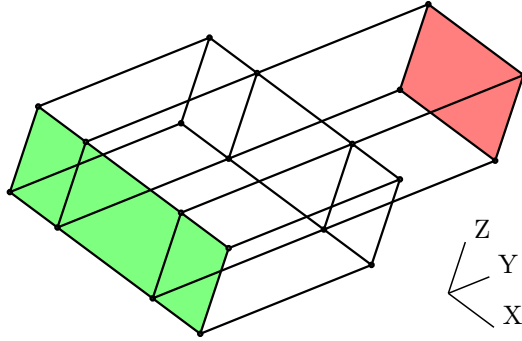


FIGURE 3.1. Geometry of an example structure for an FEM simulation where one side of the structure (green) is clamped and on the opposite side (red) a load is applied.

be used to provide information about the occurring mechanical forces in an operating load case. In this case, the FEM simulation can determine a solution that describes the resulting deformation of the structure. Further boundary conditions can be set to prevent that specific parts are deformed, for example, to simulate a clamped structure.

Figure 3.1 illustrates an example structure consisting of two connected cuboids with different size in x-direction. The corresponding mesh consists of 20 vertices, 36 edges, 21 faces and 4 elements. This example structure is used in the following to demonstrate the different FEM data formats. The boundary conditions are chosen such that one side of the structure (green) is clamped and on the opposite side of the structure (red) a load is applied in z-direction.

**3.2. SPC-FEM format.** The SPC-FEM code [5] is an in-house development of an adaptive FEM solver. The content of the corresponding input format can be separated into parts for geometry, material, and boundary conditions. Furthermore, the SPC-FEM format contains a header with additional information, such as the degrees of freedom to be used. In general, the format uses a # character followed by a keyword to start the input data of the different parts.

The geometry part contains nodes, called vertices. After the line starting with the keyword `VERTEX`, each vertex is given by an index and its coordinates in a three dimensional space. Edges are listed after the line starting with the keyword `EDGE`. Each edge is given by an index, a type (e.g., a straight line), and the two indices of the corresponding vertices. Faces are listed after the line starting with the keyword `FACE`. Each face is given by an index, a type (e.g., a plane face), and the number of associated edges followed by the indices of these edges. Elements, called solids, are listed after the line starting with the keyword `SOLID`. Each element is given by an index, a type that specifies its material, and the number of associated faces followed by the indices of these faces. Materials are defined after the line starting with the keyword `MATERIAL`. Each single material definition comprises an index and the number of material parameters followed by the parameters to describe the material.

The SPC-FEM format supports two different types of boundary conditions. The first type of boundary conditions represents Dirichlet boundary conditions which are given after the keyword `DIRICHLET`. Dirichlet boundary conditions can be used to pre-define some values of the solution, e.g. to fix the position of some nodes. The second type of boundary conditions represents Neumann boundary conditions which are given after the keyword `NEUMANN`. Neumann boundary conditions can be used, for example,

```

#VERSION: 2.0
#DATE: Monday, December 19, 2016
#DIMENSION: 3D
#EQN_TYPE: any equation
#DEG_OF_FREE: 3
#HEADER: 9
  20  36  21  4  0  3  1  1  0
#VERTEX: 20
  1  0.5  0.0  0.0
  :
  20  0.375  0.0  0.2
#EDGE: 36
  1  1  1  2
  :
  36  1  13  16
#FACE: 21
  1  1  4  1  22  11  21
  :
  21  1  4  13  14  15  36
#SOLID: 4
  1  1  6  1  2  11  10  14  18
  :
  4  1  6  3  4  5  12  17  21
#MATERIAL: 1
  1  5  10000  0.30  0  0  0
#DIRICHLET: 3
  8
    1  0.0
    1  0.0
    1  0.0
    :
#NEUMANN: 1
  4
    1  0.0
    1  0.0
    1  10.0

```

```

/prep7

et, 1, 185

mp, ex, 1, 10000
mp, prxy, 1, 0.3

n, 1, 0.5, 0.0, 0.0
:
n, 20, 0.375, 0.0, 0.2

e, 1, 2, 3, 10, 11, 12, 13, 20
:
e, 3, 4, 5, 6, 13, 14, 15, 16

d, 8, UX, 0.0
d, 8, UY, 0.0
d, 8, UZ, 0.0
:
d, 19, UX, 0.0
d, 19, UY, 0.0
d, 19, UZ, 0.0

f, 4, FZ, 10.0
:
f, 15, FZ, 10.0

/solu
solve
finish

```

FIGURE 3.2. Example of FEM input data in the SPC-FEM format (left) and in the MAPDL format (right).

to define loads on specific parts of the structure. Each boundary condition refers to a face that is given by its index followed by one line for each degree of freedom. Each of these lines contains a type that specifies how the boundary condition is represented (e.g., with constant values) followed by the specific data of this representation.

Figure 3.2 (left) shows an example of FEM input data in the SPC-FEM format. The geometry part contains vertices (i.e., nodes), edges, faces, and solids (i.e., elements) as shown for the example structure in Fig. 3.1. The Dirichlet boundary conditions fix the coordinates of the faces of one side of the structure (e.g., face with index 8) and the Neumann boundary conditions define a load in the z-direction on the face of the opposite side of the structure (i.e., face with index 4).

**3.3. MAPDL format.** ANSYS is a commercial engineering analysis software including ANSYS Mechanical, a tool for FEM analysis. The Mechanical ANSYS Parametric Design Language (MAPDL) [2] is a scripting language that is used to describe the input data of ANSYS Mechanical. The MAPDL format can be used to define the mesh of the structure to be simulated, the properties of the materials to be

used, and the boundary conditions for the desired solution. Additionally, instructions for preprocessing steps and for the solution method can be given.

The mesh of the structure to be simulated can be described by nodes and elements. The ANSYS-FEM code supports different types of elements with various shapes and material properties. The command `et` is used to select the current element type that is used for all following definitions of elements. The material properties of the elements are defined with the command `mp` followed by specific material parameters, such as the elastic modulus `ex` and the Poisson ratio `prxy`. After the current element properties are defined, the mesh of the structure is given as nodes and elements. A node is defined with the command `n` followed by an index and its coordinates in a three dimensional space. Elements represent collections of nodes and are defined with the command `e` followed by the indices of the nodes. With the MAPDL format, the boundary conditions can be defined for nodes. The command `d` is used to define displacement constraints of nodes and the command `f` is used to define force loads at nodes. Both commands require to specify the index of the node, a label that identifies the degree of freedom, and the value to be set.

Figure 3.2 (right) shows an example of FEM input data in the MAPDL format that corresponds to the FEM input data in the SPC-FEM format presented in Fig. 3.2 (left). The FEM input data begins by calling the preprocessor. Afterwards, the element type and the material properties are set and the nodes and elements are created. The boundary conditions are defined, by setting the displacement of the node with index 8 (and several other nodes) and by setting a force load in the z-direction for the node with index 4 (and several other nodes). Finally, the solver is started with the command `solve` and after finishing the computations, the ANSYS-FEM code is terminated with the command `finish`.

**3.4. SPC-FEM to MAPDL format conversion.** For parsing the source format of the FEM input data, we have used the Pyparsing library [16]. Pyparsing is an open source Python module for creating grammars and parsing text according to those grammars. For each FEM data format, a separate grammar has to be created. The given FEM input data is then parsed with the grammar of the source format.

Figure 3.3 shows a Python code example that defines the Pyparsing grammar for the SPC-FEM format. The grammar is build up hierarchically by defining separate parsing objects for each entry of the SPC-FEM format. Comment lines starting with the literal `##` are matched as entries whose occurrence is ignored. Commonly occurring entries, such as numbers, are defined once with a parsing object (e.g., `inum` and `fnum`) and then used in the definition of further entries. The definition of each specific FEM data entry uses its keyword as a literal and a corresponding parsing action that is executed when the entry is matched by the parser. The utilised parsing actions (e.g. `fill_info` or `fill_data_vertex`) store the parsed data in a dictionary-based data structure. Finally, the entire grammar object `spc_data` is defined by listing all previously defined entries that can occur within the source format.

After parsing the FEM input data given in the SPC-FEM format, the corresponding MAPDL format has to be generated. Since the two formats can use different representations for the mesh of the structure, an appropriate data transformation has to be applied. For example, the MAPDL format defines elements as collections of nodes, whereas the SPC-FEM format defines elements based on faces. Converting an element from the SPC-FEM format into the MAPDL format thus requires to determine all nodes of the element. For each element, the corresponding data transformation iterates over all participating faces, edges, and vertices to build the required collec-



```

comment = Suppress(Literal("##")+restOfLine)

inum = Regex(r' -?\d+')
fnum = Regex(r' -?\d+(\.\d*)?([eE] -?\d+)?')

config_keys = oneOf("VERSION DATE DIMENSION EQN_TYPE DEG_OF_FREE ...")
config_info = Group(Suppress("#")+config_keys+Suppress(": ")
                    +restOfLine).setParseAction(fill_info)

header_info = Group(Suppress("#")+Literal("HEADER")+Suppress(": ")
                    +restOfLine).setParseAction(fill_info)
header_data = Group(OneOrMore(inum)).setParseAction(fill_data_header)

vertex_info = Group(Suppress("#")+Literal("VERTEX")+Suppress(": ")
                    +restOfLine).setParseAction(fill_info)
vertex_data = Group(OneOrMore(fnum)).setParseAction(fill_data_vertex)

...

spc_data = OneOrMore(comment+lineEnd
                    | config_info+lineEnd
                    | header_info+lineEnd+LineStart()+header_data+lineEnd
                    | vertex_info+lineEnd+LineStart()+vertex_data+lineEnd
                    | ...)

```

FIGURE 3.3. Definition of the Pyparsing grammar for the SPC-FEM format.

tion of nodes. The resulting information is also stored within the dictionary-based data structure. After all data transformations are performed, the MAPDL format is generated by iterating over the required entries of the dictionary-based data structure.

**3.5. Flexible FEM data conversions.** The conversion of FEM data formats described in the previous subsections is closely tied to the SPC-FEM and MAPDL format. To provide a more flexible FEM data conversion for the complex simulation application developed in Section 2, we design the FEM data conversion as follows. A dictionary-based data structure is used as a central data pool that stores all information involved in the conversion. Strings are used as keys for the dictionary entries such that both format-independent and format-specific data fields can be stored. In general, a data conversion is then performed by executing the following operations:

**Source format parsing:** These operations import the given FEM input data and fill in the corresponding data fields of the central data pool. For each supported FEM data format, a separate parser operation is developed as demonstrated for the SPC-FEM format. Furthermore, parsing operations might also be implemented for importing only specific parts of the FEM input data, such as the mesh of the structure.

**Data transformation:** These operations read the existing data fields of the central data pool, translate or combine them into new information, and write them into the corresponding data fields of the central data pool. For example, determining the nodes for each element as described for the SPC-FEM to MAPDL format conversion is implemented as a transformation operation. A further example of a transformation operation is the refinement of the given mesh of the structure by dividing each element into several smaller elements.

**Target format generation:** These operations use the existing data fields of the

central data pool and export them with a specific data format. For each supported FEM data format, a separate generator operation is developed as demonstrated for the MAPDL format. Furthermore, generator operations might also be implemented for exporting only specific parts of the FEM input data, for example, to visualise the mesh of the structure with separate tools.

Performing a specific FEM data conversion is achieved by flexibly composing the required operations. The usage of a dictionary-based data structure avoids any limitations that might exist in specific FEM data formats. Additional FEM data formats can be easily integrated by providing the implementations of the corresponding parser or generator operations. The memory requirements of the conversion are mainly determined by the size of the dictionary-based data structure where the geometry information usually represent the largest entries. Currently, all operations of the conversion are performed sequentially and one after another. While parsing the source format and generating the target format are inherently sequential operations, data transformations might also be performed in parallel.

**4. Component-based implementation for distributed systems.** The individual software components of the complex simulation application for the optimisation of lightweight structures need to be executed on dedicated hardware platforms, such as desktop computers, HPC clusters, or storage servers. We utilise the Simulation Component and Data Coupling (SCDC) library to implement the interactions between these components. In the following, we give an overview of the SCDC library and describe their usage for implementing the client and service components described in Section 2.2. A more detailed description of the SCDC library is given in [12].

**4.1. Simulation Component and Data Coupling (SCDC) library.** The SCDC library is a programming library that can be used by an application programmer to implement service-oriented interactions between client and service components. In general, the interactions supported by the SCDC library are application-independent and proceed according to the following scheme: A service component provides access to *datasets* that are managed by *data providers*. A client component interacts with service components by executing *commands* on their provided datasets. The execution of a command allows to transfer input data from the client to the service and output data from the service to the client. The SCDC library provides a C and a Python interface for its library functions. Service-side functions are used to set up the data providers, to configure the data exchange methods to be used for data transfers, and to keep a service component running. Client-side functions are used to execute commands on datasets.

The datasets of an SCDC service are identified with an URI-based addressing scheme. This address identifies the specific SCDC service to interact with, the data access method to be used for data exchanges, and the specific dataset to be accessed. The following data exchange methods are supported: direct function calls between components of a single process, inter-process communication with Unix Domain Sockets between components in separate processes of a single host system, and network communication with TCP/IP sockets between components in separate host systems. The implementation details of these different data access methods are hidden in the SCDC library. Thus, an application can easily switch between different components and their specific data access methods without additional programming efforts.

The functionalities of datasets and commands depend on their specific data providers. The SCDC library contains data providers with pre-defined functionalities as well as with functionalities that can be defined by the programmer. In this

work, the following data providers are used:

- The **jobrun** data provider implements a job-oriented execution of arbitrary programs. Datasets represent the jobs to be executed and commands are used to submit jobs with their input data as well as to wait for the completion of jobs and to retrieve their output data. The program to be executed for each job has to be defined by the programmer either as a shell command or as a handler function. A list of hosts can be provided for executing the jobs. The assignment of jobs to hosts is either performed in a round-robin way or according to a determined schedule.
- The **store** data provider implements a nonhierarchical folder-oriented storage within the local file system. Datasets represent the folders and the commands are used to store and retrieve the data items of the folders.
- The **hook** data provider implements a mechanism for executing arbitrary functions whenever a dataset is accessed. All functionalities of datasets and commands have to be defined by the programmer. This data provider represents a generic mechanism to set up a service with arbitrary functionality while the accessibility to the service is still handled by the SCDC library.

**4.2. Implementation of client and service components of the optimisation application.** Each software component of the complex simulation application for the optimisation of lightweight structures is implemented in Python as a separate module. Thus, it is possible to execute the components flexibly combined, for example, in a single Python program on one hardware platform or in separate Python programs on dedicated hardware platforms. Since all interactions between the software components are performed with the SCDC library, only the addresses used for accessing the service components have to be changed if their execution platforms are changed. The software components and their interactions as described in Section 2.2 are implemented with the SCDC library as follows:

- The **optimisation** component is implemented as a client that is started by the user of the simulation application. The SCDC library is used to execute commands for storing the FEM input data of the FEM simulation tasks on the storage component, for submitting FEM simulation tasks to the generic FEM component, and for retrieving the result data of the FEM simulation tasks from the storage component.
- The **generic FEM** component is implemented as both a service and a client. The SCDC library is used to set up a jobrun data provider that executes the submitted FEM simulation tasks as jobs. The program to be executed for each job is defined by a handler function that uses the SCDC library as a client to execute further commands. These commands access the storage component to retrieve the FEM input data, the conversion component to perform the conversion of the FEM data formats, and the SPC-FEM or ANSYS-FEM component to execute the FEM simulation task with the requested FEM code. Additionally, the SCDC library is also used as a client to execute commands that access the scheduling component to determine a schedule for the execution of the FEM simulation tasks.
- The **SPC-FEM** and **ANSYS-FEM** components are both implemented as services. The SCDC library is used to set them up with jobrun data providers that execute the submitted FEM simulation tasks as jobs with the corresponding FEM code configured as a shell command.
- The **scheduling** component is implemented as a service. The SCDC library

is used to set up a hook data provider that computes a schedule when an accessing client requests it by executing a corresponding command. Command parameters are used to select a specific scheduling method. The input data of the command contains the information about the scheduling problem and the output data returns the determined schedule.

- The **conversion** component is implemented as a service. The SCDC library is used to set up a hook data provider that performs a data conversion when an accessing client requests it by executing a corresponding command. Command parameters are used to select the specific parser, data transformation, and generator operations as described in Section 3.5. The input data of the command contains the FEM input data in the source format and the output data returns the converted result in the target format.
- The **storage** component is implemented as a service. The SCDC library is used to set up a storage data provider that stores the data within a configured folder of the local file system.

**5. Performance results.** In this section, we present performance results of the FEM data conversion described in Section 3 and investigate the resulting overhead in comparison to the FEM codes.

**5.1. Experimental setup.** The measurements are performed on a single compute node with a 4-core Intel Core i5-4440 processor with 3.10 GHz, 8 GB main memory, and a 240 GB solid-state drive. The FEM data conversion is implemented as a Python module and uses files in the local file system for reading and writing the FEM input data with the different data formats. The SPC-FEM code is an in-house FEM program implemented in Fortran that performs adaptive mesh refinement based on residual type error indicators to achieve high precision solutions. The ANSYS-FEM code is part of the commercial software package ANSYS Workbench, version 16.2. Both FEM codes use multithreading to exploit the available cores of the compute node. Thus, only shared-memory parallelism is used in the experiments. All software programs are executed under the Scientific Linux 7 (64-bit) operating system and using Python interpreter of version 2.7.5.

**5.2. Data conversion performance and overhead.** The example structure shown in Section 3 is used as FEM input data for the following benchmark measurements. The original mesh of the structure consists of four elements. A refinement operation that subdivides each element into eight smaller elements has been used to increase the data sizes. Thus, with four refinement steps, the original mesh with four elements is refined into geometries with 32, 256, 2048, and 16384 elements. The data conversion is performed from the SPC-FEM format to the ANSYS-FEM format. Table 5.1 lists the resulting file sizes with the two formats.

Figure 5.1 (left) shows runtimes for parsing the source format and generating the target format during the FEM data conversion depending on the number of elements in the FEM input data. The results show that parsing the source format requires significantly more runtime than generating the target format. The high computational costs of the Pyparsing module correspond to about 80–90% of the overall runtime for the FEM data conversion. The runtimes of both operations depend on the size of the mesh and, thus, increase strongly for increasing numbers of elements.

Figure 5.1 (right) shows runtimes for the FEM data conversion as well as for executing the SPC-FEM and ANSYS-FEM codes depending on the number of elements in the FEM input data. The overhead of the FEM data conversion in comparison to

Number of elements	SPC-FEM	ANSYS-FEM
4	1.5	1.6
32	7.6	4.0
256	54.4	22.6
2048	454.4	175.8
16384	3969.3	1491.1

TABLE 5.1

File sizes in KB for the FEM input data of the example structure with the SPC-FEM and ANSYS-FEM formats (without multiple white spaces).

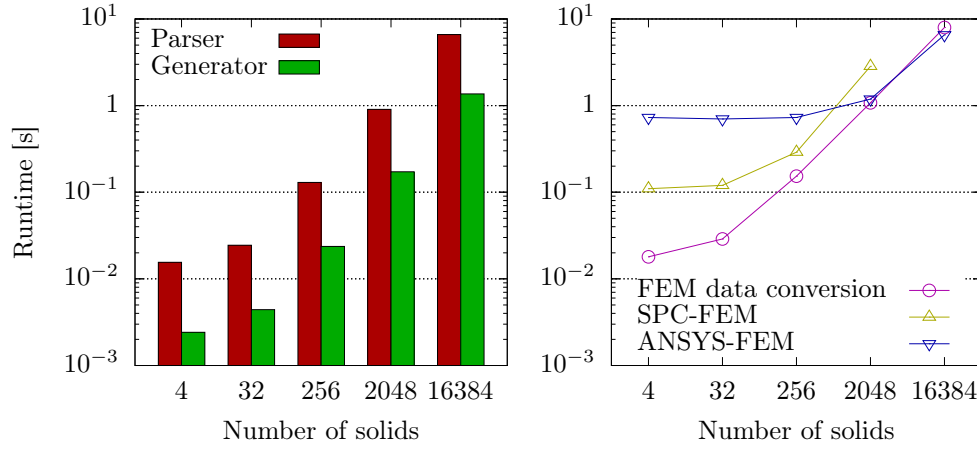


FIGURE 5.1. Runtimes of the parser and the generator for the FEM data conversion (left). Runtimes of the FEM data conversion and the SPC-FEM and ANSYS-FEM codes (right).

the FEM codes depends on the size of the mesh as well as on the specific FEM code. Up to about 256 elements, the runtime of the FEM data conversion is significantly lower and does not lead to a noticeable overhead. However, with 2048 and more elements, the runtime of the FEM data conversion increases strongly and is about equal or even higher than the runtime of the ANSYS-FEM code. As already shown, this high overhead is mainly caused by the parsing of the source format with the Pyparsing module. A direct comparison of the runtimes of the two FEM codes demonstrates their different approaches: The SPC-FEM code has a low overhead and is faster for small geometries. These are the preferred use cases, because its adaptive method is designed for starting with a coarse mesh and then refining the mesh adaptively where it is necessary. Results with 16384 elements could not be obtained for the SPC-FEM code, because an initial mesh of this size is not supported.

**6. Related work.** In scientific computing, the interchangeability of single software components represents a widely used concept that is especially used for compute-intensive and numerical computations. Being able to switch between different software components is used to achieve different goals. For standard libraries, such as BLAS [6] or LAPACK [1] for linear algebra computations, various implementations that are specially adapted to specific hardware platforms exist. These different implementations represent interchangeable software components that can be used to achieve performance improvements. Domain-specific libraries, such as the parallel graph algorithm

library PT-SCOTCH [7] or the ScaFaCoS library for the computation of Coulomb interactions [3], provide different solution methods. These different methods represent interchangeable software components that can be used, for example, to exploit the properties of specific solution methods or to improve the accuracy of the results. The flexibility achieved with these approaches is based on the fixed programming interface of the software libraries and, thus, requires that the functional properties of all software components are the same. The approach proposed in this work is not limited to such a fixed functionality, because the individual composition of arbitrary transformation operations allows more flexible and extensible data conversions.

Environmental research is one of the most prominent areas for the development of complex simulation applications, as it involves a variety of models from different disciplines, such as atmospheric sciences, hydrology, geology, chemistry, and ecology. These applications often consist of independently developed software components for the different models. The interoperability and therefore also the interchangeability of the software components is achieved through the usage of common frameworks and toolkits. For high performance computing, this includes, for example, the Earth System Modelling Framework [11], the Common Component Architecture [4], and the Model Coupling Toolkit [14]. These approaches require that all interchangeable components implement the same interface, which often involves additional programming efforts as well as a limitation of the component functionalities. In comparison, our component-based approach is less invasive to existing application codes and allows to flexibly exploit the varying functionalities of different FEM codes.

Performing explicit data conversions as proposed in this work might also be achieved with dedicated data conversion tools. However, these tools usually support only specific parts of FEM data formats, such as the geometry of the structure. For example, the mesh generator Gmsh [9] supports the conversion between different mesh formats. The parser codes of these tools are either manually constructed or automatically created with parser generators, such as Lex and Yacc [15]. For standard formats, such as XML or VTK, existing programming libraries, such as libxml [18] or the Visualization Toolkit [17], can be used for parsing the source format or even for generating the target format. The proposed approach for the FEM data conversion can incorporate these existing tools and libraries for the implementation of specific parser, data transformation, or generator operations.

**7. Conclusion.** We have demonstrated that an integration of generic FEM simulations can be performed with different FEM codes and built into a complex simulation application. A component-based approach was presented to achieve a flexible implementation with interchangeable software components. A dedicated conversion component was developed and the FEM data conversion between data formats of two specific FEM codes was shown. The proposed method allows a flexible data conversion based on the composition of individual operations for parsing the source format, performing additional data transformations, and generating the target format. We demonstrated the overall approach with a complex simulation application for the simulation-based optimisation of lightweight structures. Performance results were shown to investigate the computational effort for the FEM data conversion and to compare their overhead with the FEM codes. The results have shown that especially the parsing of the source format can lead to a high overhead. Thus, it is highly required to provide a flexible data conversion approach where single operations can be easily replaced or optimised.

**Acknowledgements.** 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

- [1] E. ANDERSON, Z. BAI, C. BISCHOF, S. BLACKFORD, J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORESENSEN, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, 3rd ed., 1999.
- [2] ANSYS, INC., *ANSYS Parametric Design Language Guide*, 2015.
- [3] A. ARNOLD, F. FAHRENBERGER, C. HOLM, O. LENZ, M. BOLTEN, H. DACHSEL, R. HALVER, I. KABADSHOW, F. GÄHLER, F. HEBER, J. ISERINGHAUSEN, M. HOFMANN, M. PIPPIG, D. POTTS, AND G. SUTMANN, *Comparison of scalable fast methods for long-range interactions*, Physical Review E, 88 (2013), p. 063308.
- [4] D. BERNHOLDT, B. ALLAN, R. ARMSTRONG, F. BERTRAND, K. CHIU, T. DAHLGREN, K. DAMEVSKI, W. ELWASIF, T. EPPERLY, M. GOVINDARAJU, D. KATZ, J. KOHL, M. KRISHNAN, G. KUMFERT, J. LARSON, S. LEFANTZI, M. LEWIS, A. MALONY, L. MCINNEN, J. NIEPLOCHA, B. NORRIS, S. PARKER, J. RAY, S. SHENDE, T. WINDUS, AND S. ZHOU, *A component architecture for high-performance scientific computing*, Int. J. of High Performance Computing Applications, 20 (2006), pp. 163–202.
- [5] S. BEUCHLER, A. MEYER, AND M. PESTER, *SPC-PM3AdH v1.0 - Programmer's manual*, Preprint SFB/393 01-08, TU-Chemnitz, (2001).
- [6] L. BLACKFORD, J. DEMMEL, J. DONGARRA, I. DUFF, S. HAMMARLING, G. HENRY, M. HEROUX, L. KAUFMAN, A. LUMSDAINE, A. PETITET, R. POZO, K. REMINGTON, AND R. WHALEY, *An updated set of basic linear algebra subprograms (BLAS)*, ACM Transactions on Mathematical Software, 28 (2002), pp. 135–151.
- [7] C. CHEVALIER AND F. PELLEGRINI, *PT-Scotch: A tool for efficient parallel graph ordering*, Parallel Computing, 34 (2008), pp. 318–331.
- [8] R. DIETZE, M. HOFMANN, AND G. RÜNGER, *Water-level scheduling for parallel tasks in compute-intensive application components*, J. of Supercomputing, Special Issue on Sustainability on Ultrascale Computing Systems and Applications, 72 (2016), pp. 4047–4068.
- [9] C. GEUZAIN AND J.-F. REMACLE, *Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities*, Int. J. for Numerical Methods in Engineering, 79 (2009), pp. 1309–1331.
- [10] S. HANNUSCH, R. HERZOG, M. HOFMANN, J. IHLEMANN, L. KROLL, A. MEYER, F. OSPALD, G. RÜNGER, R. SPRINGER, M. STOCKMANN, AND L. ULKE-WINTER, *Efficient simulation, optimization, and validation of lightweight structures*, in Proceedings of the 2nd Int. MERGE Technologies Conference for Lightweight Structures (IMTC 2015), Verlag Wissenschaftliche Scripten, 2015, pp. 219–227.
- [11] C. HILL, C. DELUCA, V. BALAJI, M. SUAREZ, AND A. DA SILVA, *The architecture of the earth system modeling framework*, Computing in Science & Engineering, 6 (2004), pp. 18–28.
- [12] M. HOFMANN AND G. RÜNGER, *Sustainability through flexibility: Building complex simulation programs for distributed computing systems*, Simulation Modelling Practice and Theory, Special Issue on Techniques And Applications For Sustainable Ultrascale Computing Systems, 58 (2015), pp. 65–78.
- [13] J. KLEIJNEN, W. VAN BEERS, AND I. VAN NIEUWENHUYSE, *Expected improvement in efficient global optimization through bootstrapped Kriging*, J. of Global Optimization, 54 (2012), pp. 59–73.
- [14] J. LARSON, R. JACOB, AND E. ONG, *The Model Coupling Toolkit: A new Fortran90 toolkit for building multiphysics parallel coupled models*, Int. J. of High Performance Computing Applications, 19 (2005), pp. 277–292.
- [15] J. LEVINE, T. MASON, AND D. BROWN, *lex & yacc*, O'Reilly & Associates, Inc., 2nd ed., 1992.
- [16] P. MCGUIRE, *Getting Started with Pyparsing*, O'Reilly Media, Inc., 2007.
- [17] W. SCHROEDER, K. MARTIN, AND B. LORENSEN, *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*, Kitware, 2006.
- [18] D. VEILLARD, *libxml – The XML C parser and toolkit of Gnome*.