Open Source Software Framework for Applications in Aeronautics and Space

Doreen Seider
Distributed Systems and Component Software
Simulation and Software Technology
German Aerospace Center (DLR)
51147 Cologne, Germany
Doreen.Seider@dlr.de

Philipp M. Fischer
Software for Space Systems and Interactive Visualization
Simulation and Software Technology
German Aerospace Center (DLR)
38108 Braunschweig, Germany
Philipp.Fischer@dlr.de

Markus Litz and Andreas Schreiber
Distributed Systems and Component Software
Simulation and Software Technology
German Aerospace Center (DLR)
51147 Cologne, Germany
Markus.Litz@dlr.de
Andreas.Schreiber@dlr.de

Andreas Gerndt
Software for Space Systems and Interactive Visualization
Simulation and Software Technology
German Aerospace Center (DLR)
38108 Braunschweig, Germany
Andreas.Gerndt@dlr.de

***Abstract*—The DLR developed the open source software framework RCE to support the collaborative and distributed work in the shipyard industry. From a technology side of view a software from the shipbuilding field has many requirements in common with aerospace software projects. Accordingly, RCE has become the basis for further projects within the DLR. Over the last years of usage a subset of frequently used software components could be derived and are provided by the RCE framework. In particular, the workflow engine, allowing the integration of different domain-specific tools from local and remote locations into one overall calculation has become important for various projects. We present RCE and show how its software components are reused in two aerospace applications.**

TABLE OF CONTENTS

1. INTRODUCTION

The German Aerospace Center (DLR) together with Fraunhofer and the shipyard industry developed a domain-specific design and simulation environment called SESIS [1]. Looking to such industry shows that designing a ship requires knowledge of various experts and engineers from very different domains. All the data and work they contribute towards the final product, strongly depends on each other. For example, the fluid dynamics of a ship's body influences its efficiency concerning fuel consumption, which as a consequence influences the design of fuel tanks and engines used for such a vehicle. With this in mind, the aim of SESIS was to support engineers with software, allowing them to contribute into the same design by sharing data across disciplines. Even more, it was intended to offer a software capable of connecting domain-specific tools of the shipyard industry considering their dependencies and of contribution into one global calculation bringing results for the intended design.

With the beginning of the project some of the requirements concerning the integration of such domain-specific tools requested that these tools can be connected via a network running on different platforms. The reason behind this requirement was, that many domain-specific tools can either only be controlled by their respective engineers, need to run on a specific platform, or require a too expensive license to allow an installation on various computers. Accordingly, for SESIS distribution was inevitable. TENT [2], a former DLR project was already fulfilling such distribution requirements, but it was outdated due to its underlying architecture and technology. As a consequence, the project *Remote Component Environment* (RCE) [3] was initiated to create a new framework incorporating the lessons learned from TENT and being based on a new and modern software architecture. But different to TENT which used to be an integration framework solely, RCE intended to be more generic and to enable reuse for various future software projects. By using a component-based approach, future applications can reuse needed software components to reduce development costs and time. With proposing and implementing such a framework, already fulfilling the demands on distribution infrastructure, data management and further important components, SESIS could be easily finalized on top.

Comparing the shipyard industry with aeronautics and space, they have a lot in common. All of them develop and design complex systems with knowledge of various very different disciplines. But same as in the shipyard industry, all of them have strong dependencies to each other. A brief look to the space domain depicts, that the overall mass of the system influences the launcher that can take it into space. In return, the launcher constrains the overall structure of the spacecraft creating further implications to other domains. All of the experts being involved in the design of such systems use domain specific tools for their calculations as well. And they use and provide data from and to other disciplines. Thus, SESIS, as a supporting software in research and development in the shipbuilding field, has many requirements in common with software we develop in aerospace fields at DLR [4].

978-1-4577-0557-1/12/$26.00

In the following we show requirements we identified as main design and development driver of new software in such complex engineering and science fields. By the support of RCE's component-based approach it lends itself as a reasonable framework for such applications. Accordingly, we explain how RCE that was initially built for the shipyard industry still fits to these requirements, by giving a proof of concept referring to two real life examples: *Chameleon*, an environment for the scientific predesign of aircrafts and *Virtual Satellite*, a supportive application for concurrent engineering sessions in the spacecraft domain.

In Section 2 we introduce the requirements of the component-based framework approach mentioned above. The actual RCE software framework and the software components it is offering for reuse are described in Section 3. Afterwards, in Section 4 and 5 we focus on Chameleon and Virtual Satellite, two applications from aerospace we realized on top of RCE. They serve as proof of concept examples. Thereby, we highlight the benefits of reusing RCE's software components. Finally, we give a conclusion in Section 6.

2. REQUIREMENTS FOR THE SOFTWARE FRAMEWORK

As mentioned, reuse was a major concern for the development of RCE as it was intended by SESIS. The environments of today's software projects are just to complex to come along not having reuse in mind [5]. This is surely also the case for all the projects that were based on RCE over the last years. By having a closer look to reuse in software engineering, often three levels of reuse are mentioned: reuse on code level, reuse on pattern level and reuse on framework level [5].

By default, frameworks abstract and provide higher level functionality for reuse. Furthermore, unlike libraries frameworks are in charge of the application's flow of control [6] often called inverse of control. Therefore, frameworks are extensible through a callback style of programming. This means contributions to a framework are done by implementing appropriate interfaces provided by the framework without being responsible for the flow control of the contributions. Thus, the framework calls the implementation instead of the implementation calls the framework. As a consequence, using frameworks demands less understanding about the whole application. This enables easy reuse and extension of the framework since developers just have to comply to the contract of the mentioned interfaces. Still these frameworks have to provide the trade-off between generalization and individuality [7]. Which means on the one hand, a framework which tries to do everything might be to be too specialized for reuse in a broad field of applications. On the other hand, a too generalized framework might not be efficient enough for reuse. Accordingly, writing a reusable framework like RCE demands to consider many aspects. NASA provides the Reuse Readiness Levels (RRLs) [8] judging software on nine criteria to estimate its reuse potential. These criteria are *documentation, extensibility, intellectual property issues, modularity, packaging, portability, standards compliance, support and verification and testing*. Even though, some of these criteria are addressing the internals of software, others like modularization, portability and extensibility are referring to aspects and requirements of RCE already mentioned in the introduction.

Reflecting these criteria for software reuse and applying them to the RCE framework, we derived the following overall requirements for it:

- The framework needs to provide a component-based approach, so that users and developers can decide on their own which parts they are intending to reuse and which parts are redundant for them.
- The framework needs to be extensible, so that developers and users can either extend the functionality of it to fulfill their individual requirements or let their domain-specific tools interact with that framework.
- The framework needs to be portable so that it can be used on a broad variety of host platforms, leaving domain specific tools in their natural environment.
- The framework needs to be free of licensing issues conflicting with the distribution demands in scientific applications.

The listed requirements affect the framework itself. They do not consider the mentioned high level functionality a framework provides. To be aware of the functionality RCE must provide, it is essential to identify mutual requirements of relevant applications built on top. Beginning with SESIS which was delivering the first requirements up to today, six mutual requirements were identified that play an essential role for software in the scientific aeronautics and space domain:

- *Distribution*: provide capability to realize distributed software.
- *Data Management*: provide management for scientific data.
- *Privilege Management*: provide user management with authentication and authorization support.
- *Workflow Engine*: provide environment for coupling of tools to automated compute workflows.
- *Graphical User Interfaces*: provide general-purpose elements.
- *Platform Independence*: executable on different operating systems.

Looking back over the past years of developing RCE, some additional "soft"-requirements evolved. It is worth to mention that the "hard"-requirements we discussed so far do not lend themselves as a guarantee for developing a reusable framework. It is mainly up to the developers using RCE as a basis for new projects that drive requirements and that finally, decide if the framework is reusable or not. So, to get acceptance by these developers the following two requirements had a major impact to the last years of RCE development:

- Provide software components which are as individual as needed and as general as possible.
- Realize a reusing concept which is the most easiest possible to apply by software developers.

In the following parts we will have a closer look to the six core components of RCE as well as their individual requirements.

Distribution

As mentioned ships, airplanes and spacecrafts are complex systems. They require shared knowledge of various different experts throughout the design and development process. Today, these contributors like suppliers, scientists, and developers are situated in various locations. Thus, providing knowledge to a system like a ship or aircraft demands to work distributed towards a common product. Concerning RCE this means to exchange data and tools across a common project environment. Reasons are a tool demanding a

special database that cannot be easily moved, or corporate governance that only allows to hand out results but no tools. In this way RCE allows contributors to access data and tools from anywhere within the project and to integrate them into their local calculations.

Hence, the RCE framework must provide features and functionalities to share such data and tools in a common environment. Moreover, it needs to enable all remaining parts of RCE and in particular domain-specific software on top of it to operate distributed.

Data Management

During research and development activities, data is generated mostly all of the time. Such data is provided distributed from various different contributors. The type of data can range from simulation results of an optimization on a cluster, over a new ship design on a shipyard's file server, up to experiment data of the latest wind tunnel experiment on a scientist's laptop. Hence, it demands different storage back-ends for such a data management. Primarily, because they might change: Data stored today on a local relational database might move to a cloud service tomorrow. Even though, the data is distributed over several different storage devices, it is often required by the work of all individuals contributing to a ship or spacecraft. If users are not able to find the proper data of their interest, it might influence research and development results, by either raising unwanted inconsistencies or redundancies. Thereby, data access issues may also be a concern to either avoid unauthorized persons of providing new data or changing it.

As a consequence, a data management needs to fulfill the requirements of providing capabilities to connect to different storage back-ends, keeping them abstracted and transparent, so that developers and users do not necessarily need to be aware of them. Furthermore, it needs to provide mechanisms to store and retrieve data of different type and size. Finally, it has to provide user interaction for browsing and searching data in accordance to the privileges of the user.

Privilege Management

In an environment where many suppliers, researchers and developers contribute into a common design, roles and privileges need to be well defined. In particular, for the reasons of sharing data in such collaborative environments, it is necessary to protect it from misuse. This does not cover high level security like encryption. In such environments its more a concern of ensuring that only qualified individuals store and alter respective data. Additionally, this concerns the usage of such data as well, because if misinterpreted, proceeding results might corrupt and influence the overall design.

Summarizing, a privilege management needs to provide means to protect data or tools against misuse. Moreover, it needs to provide other components such as data management with relevant information about individual user rights.

Workflow Engine

To work collaboratively it is necessary to connect tools such as Excel sheets, CFD (Computational Fluid Dynamics), and FEM (Finite Element Method) calculations together, this is often called process chains or workflows. Such workflows are often used in research and development [9]. The mentioned tools can be distributed within the common project environment, but still they are intended to interchange data across all RCE instances. By using such workflows connecting own local tools with remote ones, engineers can reflect their contribution with other discipline's tools and knowledge. In combination with optimizers or parametric studies, engineers can quickly evaluate ideas in the global context of the final product. To connect all these components, data mappings have to be defined telling where data is provided and where it is consumed. Such mapping can be a challenge on its own, hence, engineers need to be provided with adequate feedback and insight, e.g., logging, interim results and debugging facilities.

All this demands individual requirements for the workflow engine, such as providing engineers with a generic set of workflow components to do optimizations or parametric studies. It has to be possible to integrate tools as workflow components by either providing easy to use interfaces or generic wrapper components. An interface implemented by a developer has to allow for integrating new tools, while generic wrapper components have to provide functionalities for wrapping Excel sheets or Python scripts and executing them within a workflow. Furthermore, a workflow engine has to provide a distributed execution environment that can be configured locally by an individual engineer, to influence the dedicated execution as well as the data flow. Additionally, such an environment needs to provide local facilities to inspect and debug final and interim results from the distributed contributions.

Graphical User Interface

Usability demands that interaction is easy to learn, effective and efficient to use and enjoyable from the user's perspective. It needs to enable them to carry out their work [10]. Thus, developing individual graphical user interfaces (GUI) for future software is not possible due to their missing requirements. Nevertheless, it already can provide some general user interface for the components provided by RCE. This covers the login for the privilege management, browsers for the data management as well as an editor, logger and further tools for the workflow engine.

Hence, it is required that the framework provides a general GUI as described above to allow developers of future software to decide which parts to use. Furthermore, they need to be enabled to extend some of the GUI like the editor for the workflow engine, to display new workflow components being integrated.

Platform Independence

Due to the nature of the collaborative environment, different tools that run distributed can be based on different platforms. There might be databases running on Linux machines, notebooks being based on Mac OS, or workstations using Windows operating systems. RCE provides functionality to collaboratively work in such heterogeneous environments. From a user's perspective, the underlying platforms of RCE instances are irrelevant and accordingly abstracted.

The usage of different platforms demands a platform independent implementation of RCE and its software components. If this is impossible for individual components at least different platform-dependent implementations of them should be provided. It is further required that all distributed services and software components can interchange data on a platform independent level.

3. Software Framework RCE

The development of RCE started in 2005 under the lead of DLR. From that point onwards it has evolved into a framework which has become the basis for different projects. In this Section we focus on the technical realisation that was driven by the user requirements described in previous Section 2. We have a closer look to RCE's underlying system architecture as well as its software components.

RCE's System Architecture

Out of the user requirements for a platform independent framework, using a component-based approach as well as the demand for avoiding licensing issues, the Eclipse Rich Client Platform (RCP) [11] was chosen as foundation for the RCE framework. Looking to the aerospace and automotive industry or to academia, Eclipse has become a major framework in the development of today's research and development software. There are many reasons for it such as being implemented with Java offering a modern and freely available programming language to developers. Figure 1 shows the interaction of it within the context of the RCE framework. The bottom layer shows Eclipse RCP together with its (Open Services Gateway initiative) [12] OSGi-runtime Equinox. Using OSGi already fulfills the base requirement for the component-based approach, since it requires to encapsulate code in bundles. First, such bundles can be loaded and unloaded during runtime by the OSGi framework. Second, these bundles can provide services which are then consumable by other bundles. Such features enables developers of aerospace applications to decide which RCE bundles they want to include in their application, by individually activating them in the OSGi context. Next to the OSGi standard functionality Equinox offers services to build native GUI applications. Even though, the GUI is native it still fulfills the user requirement of being platform independent, since Equinox offers service implementations for various operating systems. RCE is based on this Eclipse RCP and OSGi layer by consuming services. Besides the GUI elements all of the RCE software components are plain OSGi bundles providing further services to the actual application layer on top. As shown in Figure 1, this approach allows to build applications using on the one hand the whole Eclipse RCP framework, and on the other hand the additional RCE features in parallel.

Building applications based on Eclipse RCP has a further advantage. Eclipse used to be an Integrated Development Environment (IDE) for the software development in the beginning, but turned over into the RCP project with the IDE based on it itself. This way it lends itself as an IDE, going beyond software engineering using RCE. All extensions which are already available for software development can now be integrated into any other application based on RCP. For example, it can have a Python development environment directly integrated in RCE. This is reasonable and handy if you build workflows with the Python wrapper component and thus, need to implement Python scripts. As third advantage it is worth mentioning that Eclipse is open source. Thus, developers do not have to spend extra money for licensing since RCE is open source as well. Accordingly, it fulfills a further important user requirement.

RCE as Open Source

At DLR, we develop software to support research and development activities or to solve research problems. In nearly all cases we do not develop software for selling purposes. Our business model for software development differs from conventional software companies. Usually, we do not get funding for software development from customers but as aid money from German Federal and Leander Authorities, European Union (EU) and other public institutions. This leads to the fact that software development at DLR is mostly done within research projects. For that reason the challenge for DLR is to convince project partners and mentioned public institutions to develop a specified software or to use and extend an existing one already developed at DLR.

The decision to change RCE into an open source project based on the Eclipse Public License (EPL) was raised and realised in 2010. Due to our special business model described above we did not apply one of the main open source business models listed in [13]. The decision for open source was made, because our experience showed that there was a higher acceptance for our software compared to commercial or closed source software in the DLR research project environment. This has two reasons: First, an open source software enables collaboration outside the research project scope. This is particularly applicable for partners from academia often suffering small budgets for licensing fees. Second, an open source software with a liberal license like the EPL allows individual software being built on top of it to be open source as well as being published commercially. Thereby, project partners have no restrictions using the developed software of a project after it has been finished. In turn, this is important for public institutions, because they like to fund projects with sustainable results with a profit for a wide range of organisations and not only the software development ones.

Since RCE became open source its acceptance in the DLR research project environment grew and it has been applied to further projects. The initial decision for providing RCE as open source software was made for funding reasons. But as the number of projects with RCE increases we will face challenges explained in [13]. For example, the RCE development group will not be able to implement all project's requirements because of missing time and domain knowledge. To manage such challenges in the future we will develop an open source strategy for RCE making use of the benefits of open source going beyond the ones described above, in particular building a community not only consuming but also contributing to RCE.

RCE's Software Components

Figure 2 shows the RCE software components that are derived from the mutual requirements. Additionally, it provides an overview of their dependencies in a layered architecture. Upper software components make use of lower ones. For example, the data management consumes services from the user privileges, notification as well as the distribution. The notification instead is just consuming services from the distribution component. The GUI component on the left may

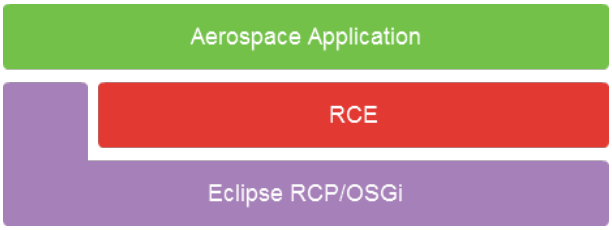

Figure 1. Interaction of RCE with Eclipse RCP/OSGi and Aerospace Application

access all of the individual components and their services.

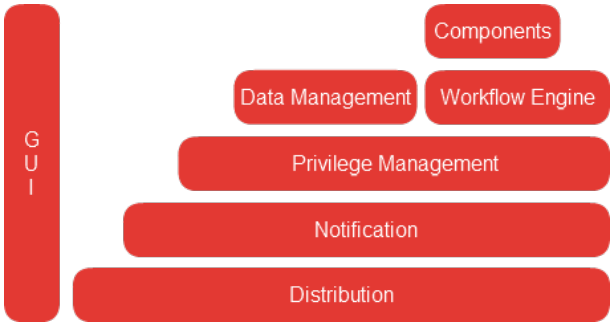

Figure 2. RCE software components and its dependencies

In the following part each of RCE's software components will be explained in more detail.

Distribution— Various RCE instances can be bundled to a common project environment. Within such RCE environments every provided service is reachable from another remote RCE instance. These remote services can be accessed from any point within the aerospace application or RCE itself. To access it, it needs to be requested from the distribution service of this component. From then on the distribution service takes care about forwarding calls to the remote RCE instance and its requested service. The supported communication protocols are RMI (Remote Method Invocation) and SOAP. Nevertheless, the component is extensible to add further protocols to meet future requirements or local network policies.

Notification—Applications with long-running tasks or with unpredictable events need a notification system. For example, it is needed to inform a workflow component of new data to be processed, or to inform a GUI element to update its content. The notification component of RCE is consuming the distribution service, accordingly, it allows for distributed notifications. Such notifications enable distributed concurrency, which is used for the data management or the workflow engine. Notification examples for such cases are 'workflow is finished', 'remote RCE instance is down', or 'query results are present' informing other RCE instances of events being important for their local control and user feedback.

Privilege Management— Privilege management consist of authentication and authorization. The first one helps to prove if someone is the person he pretends to be. The second checks for individuals' rights. Both of them are realized in RCE as services which allow to request user privileges at any point in the aerospace application or RCE itself. The privilege component consumes the notification and distribution service. This interaction allows for example to notify an RCE instance of a new user logging into the system, or to ask for tool access on a remote RCE instance.

Data Management—The data management component provides query and data storage services. The whole RCE data management is decentralized, accordingly, every RCE instance has a local data management. Since the data management is making use of the user privileges, notification and distribution service, each local data storage is accessible from any other RCE instance still obeying user rights. To store and find data appropriately, RCE annotates this data with meta tags. Users can search for these meta tags from anywhere in a common project environment, thus letting the decentralised data management appear as a whole one. So far, the data management just provides file storage solutions, but it is extensible similar to the distribution component. Accordingly, further storage types can be added at a later stage to meet individual requirements of applications on top.

Workflow Engine—The workflow engine in RCE is responsible for tool integration support and management of workflows. Workflow in RCE means integrating tools, so called workflow components and connecting them together for collaborative calculations. Workflow components are installed, managed and started locally. Each RCE instance can provide such components. The workflow engine is consuming the user privileges, notification and distribution service, thus, allowing any other RCE instances to consume and use such workflow components obeying user privileges. Nevertheless, the actual calculation and the tool used for it stays on its original machine, which means that only data is exchanged to and from such a component. Figure 3 shows such a case of a workflow managed by one local instance using two remote RCE instances contributing into a common calculation. RCE is delivered with a set of standard workflow components for parametric studies, Python execution, or Excel calculations. Accordingly, it is extensible so that individual applications can provide further components.

GUI Elements—RCE provides GUI elements which are often needed in applications built on top. The elements range from login dialog over a log browser and a data management view towards the workflow editor. Implemented in separate bundles it contributes these GUI elements to the RCP framework. Thus, they can be easily accessed, for example, a login dialog is available by traversing the standard "File"-menu provided by standard RCP workbenches or specific user applications. Having the GUI elements separated in individual bundles they are not necessarily needed to build an application on top of it. This can be important if a workflow component is needed for a tool like a database server not providing a GUI. Such a workflow component can be implemented using the core RCE software components without the GUI component, hence, providing the workflow component to other RCE instances solely as remote service.

RCE in Practice

With the described architecture and software components and the fact that RCE is written in the platform-independent programming language Java, we meet all requirements listed in Section 2. Nevertheless, a real proof of concept are the applications built on top of it.

Until today applications from different domains are built on top of RCE. The first one was the ship and simulation system SESIS. Applications in aeronautics and space have followed. These are Chameleon, an environment for the scientific predesign of aircrafts and Virtual Satellite, a supportive application for concurrent engineering sessions in the spacecraft domain. In the following we will explain how we reused RCE's base software components within these two examples, and how specified components of both of them are realized on top of RCE.

4. Application in Aeronautics: Chameleon

The DLR develop Chameleon for support of multi-disciplinary projects in aeronautic fields. Similar to the SESIS project, Chameleon needs facilities allowing differ-

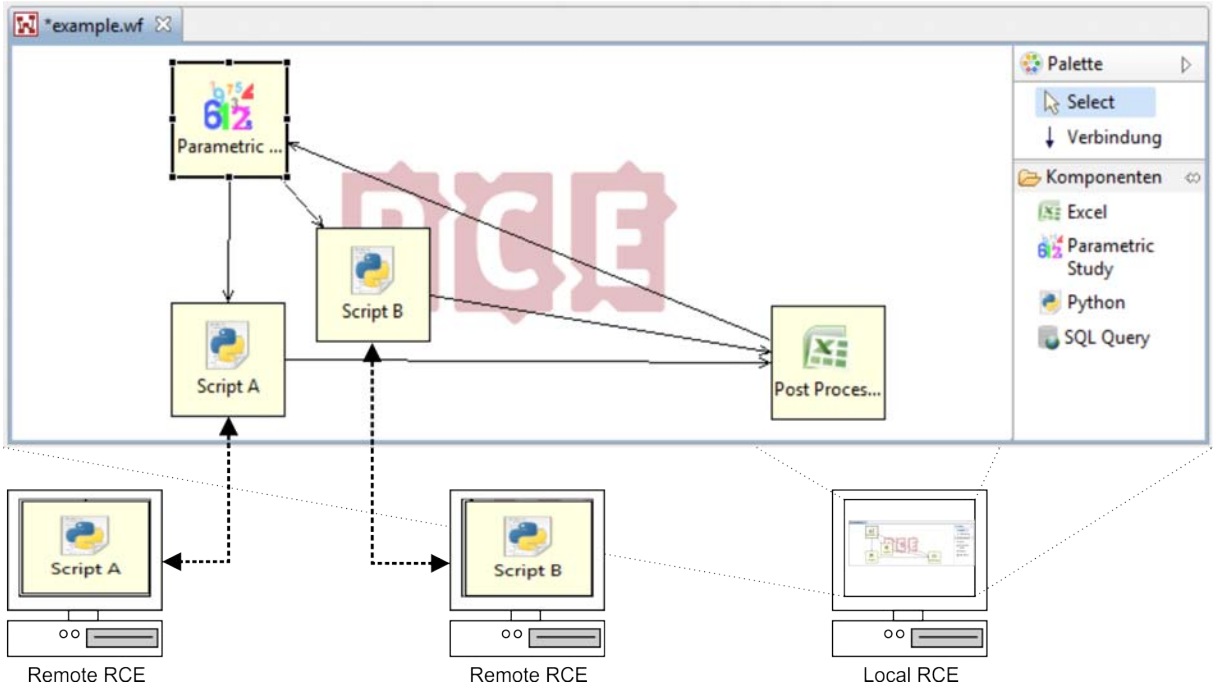

Figure 3. Example of a workflow being executed within a common project environment. The local RCE instance is managing a running the whole workflow, while the two python scripts are remotely contributing to the collaborative result.

ent contributors to work towards a common design of new aircrafts. Chameleon focusses on collaborative simulations of preliminary aircraft designs. Accordingly, the various tools used and their dependencies have to be considered and understood. Connecting them together into an overall workflow allows engineers to investigate design and ideas taking multi-disciplinary knowledge into account.

Since the beginning of Chameleon it has been applied to several different multi-disciplinary projects within the DLR. Below is an overview of these projects with a short description of their individual goals:

- *Technology integration for the virtual aircraft (TIVA I + II, VAMP)*: This project's goal was to do technology assessments regarding collaborative predesign of civil aircrafts.
- *Climate optimized air transport system (CATS)*: This project intended to optimize aircraft and missions with regards to their climatic impact.
- *Evaluation of innovative aero-engines (EVITA)*: This project focused on the preliminary design of engines rather than on the whole aircraft.
- *Unmanned Combat Air Vehicle (UCAV-2010/FaUSST)*: This project aimed to simulate the development of a military unmanned aircrafts optimized for stealth capabilities.

Reusing RCE

Due to the similarities to the project SESIS and concerning the workflows in particular, the decision was made to use an existing framework as basis for Chameleon. RCE was not ready for further use at that time, leading to an evaluation of other frameworks providing workflow services. Accordingly, the *ModelCenter* framework from Phoenix Integration [14] was chosen as base system for the early Chameleon. Unfortunately, this turned out to be a sub optimal decision for the following reasons:

- The mentioned licensing concerns decreased the acceptance of Chameleon. In particular, after projects ended, since no budget was left for new licensing fees and a continuous use and development.
- As non open source framework, it was not flexible enough to fulfill all the individual requirements from the various projects based on Chameleon.

RCE was able to overcome these drawbacks, thus it was integrated as soon as it was ready. Being open source, RCE does not suffer licensing issues and being a DLR internal project it can directly react to requirements out of the project. Thus, fulfilling individual needs which are not achievable with a commercial solution.

Figure 4 gives an overview about the architecture of Chameleon and how it is integrated in RCE. It illustrates which software components are used by Chameleon by drawing a direct connection to them. To get a better understanding why these software components are reused by Chameleon, the use cases for them are described in the following.

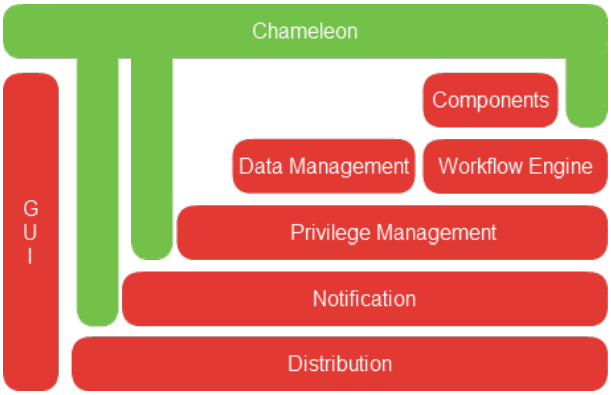

Figure 4. Chameleon architecture on top of RCE

RCE's Distribution Infrastructure—Multi-disciplinary simulations as provided by Chameleon involves tools from DLR institutes spread all over Germany. It is often required that all tools must stay at their corresponding institute for technical or organisational reasons. As a consequence, Chameleon needs to be a distributed system with the capability to execute distributed workflows. With RCE's distribution component Chameleon became a distributed simulation environment without implementing any additional code.

RCE's Notification System—Chameleon extends RCE with its own graphical user interfaces, primarily for monitoring of running tools. The notification component of RCE enables Chameleon to update the user interfaces as soon as changes occur on any relevant RCE instance within a project environment.

RCE's Workflow Engine and Components—The key component for Chameleon is RCE's workflow engine and its workflow components. With them, Chameleon is able to automatically execute simulation tools as workflows and thus, allowing for multi-disciplinary simulations. Next to standard workflow components like a parametric study or optimizer, Chameleon makes use of the provided Python component. Hence, engineers can provide simulation logic ad-hoc and on their own just by writing a Python script. Within such scripts input values from other tools are transparently accessible and output values are declared as Python variables. So, the Python component allows even the engineer to extend the framework with functionality.

A screenshot of Chameleon shows Figure 5. In the middle is the workflow editor for creating, managing and starting workflows. On the right hand side next to it is a monitoring view. It shows an investigated parameter studied by the parametric study workflow component. At the lower right hand side is the integrated TIGLViewer [15]. This viewer allows for visualization of aircraft geometries. The workflow console browser can be seen to the left hand side next to the TIGLViewer. It provides engineers with natural console output of the integrated tools.

Further Software Components of Chameleon

Every tool used in these multi-disciplinary projects expects its own data format for input and output. This needs to be taken into account for each new workflow component integrating such tools. Chameleon is used in several projects, thus, integrating a large amount of different tools. Hence, solving the input and output issues for each tool individually would clearly result in an unmanageable effort.

As a consequence, Chameleon uses a common integration concept. The backbone is a common data format called CPACS (Common Parametric Aircraft Configuration Scheme) [16]. Every tool gets its input from CPACS and writes its output back to it. Libraries, such as TIXI [17] and TIGL [15] support interaction between the tools and the CPACS format. Thus, they also support the interaction between RCE and CPACS. Accordingly, Chameleon extends RCE with workflow components supporting this common integration concept instead of individual ones for each tool.

Upshot

Chameleon is a simulation environment for preliminary aircraft design. It provides a distributed infrastructure within the DLR for multi-disciplinary simulation in aeronautics. Therefore, Chameleon focuses on aeronautic specific features implemented in the supportive components based on CPACS. Necessary features such as the distribution are provided by RCE without any further effort needed by Chameleon's developers. This reuse of RCE components is an important aspect that allows Chameleon to be rapidly developed and extended.

5. APPLICATION IN SPACE: VIRTUAL SATELLITE

In 2008 the DLR initiated a national satellite program. Aim of this program is to gain an independent access to space for scientific experiments. As part of this initiative a new Concurrent Engineering Facility (CEF) was built. To support the work in the CEF the development of the software Virtual Satellite was started as well [18].

The Concurrent Engineering Facility at the DLR

The CEF built in Bremen was inspired by the Concurrent Design Facility (CDF) [19] of the European Space Agency (ESA). It is built for communication, presentation as well as scientific and concurrent work. Accordingly, it offers twelve workstations for engineers and further places for visitors and customers. All workstations are equipped with modern computer systems to assist the work of the engineers. The engineers are seated in an open semi circle with the others sitting in the middle [18]. This semi cycle allows for an easy face to face communication, which is important in the early stage discussions of space systems. Target of these early discussions is to gain a good and reasonable idea of the planned spacecraft. Important measures in these early stage studies are the weight, costs as well as data and power budgets of the system. Following a defined process the engineers can discuss their design concerning such measures. Based on quick iterations and discussing trade-offs the engineers aim for common-ground within a short period of just a few weeks [20], [21]. Nevertheless, a software together with an integrated data model is inevitable to support the concurrent engineering process [20], [19], [22].

A Design Software for the Concurrent Engineering Facility

The software Virtual Satellite is built targeting for the needs in the CEF and the early phase studies done there. Based on Eclipse RCP the software offers an integrated data model using the Eclipse Modeling Framework (EMF) [23]. The data model allows all engineers to decompose the satellite into its components. These components are stored as so called *SystemComponents* which allow to persist corresponding parameters, calculations and additional information. The data model is filled with data through the graphical user interface of the software. A Navigator is representing the stored study in a tree like hierarchy reflecting the decomposition. Each *SystemComponent* of that decomposition can be opened and edited by an editor. Such an editor supports adding and altering parameters, calculations and further relevant data [21]. To support the collaborative aspects of CEF sessions, the data model is shared through a version control system. Any change of each engineer can be committed to a central server and forwarded to all other participants in the study. Hence, every engineer contributes into one central integrated data model [20].

Integrating RCE as a New Basis

Virtual Satellite started as an experimental scientific project. As it turned out that it will pass the proof of concept phase the decision was made that Virtual Satellite will become a productive software for the use in the CEF. Therefore, new

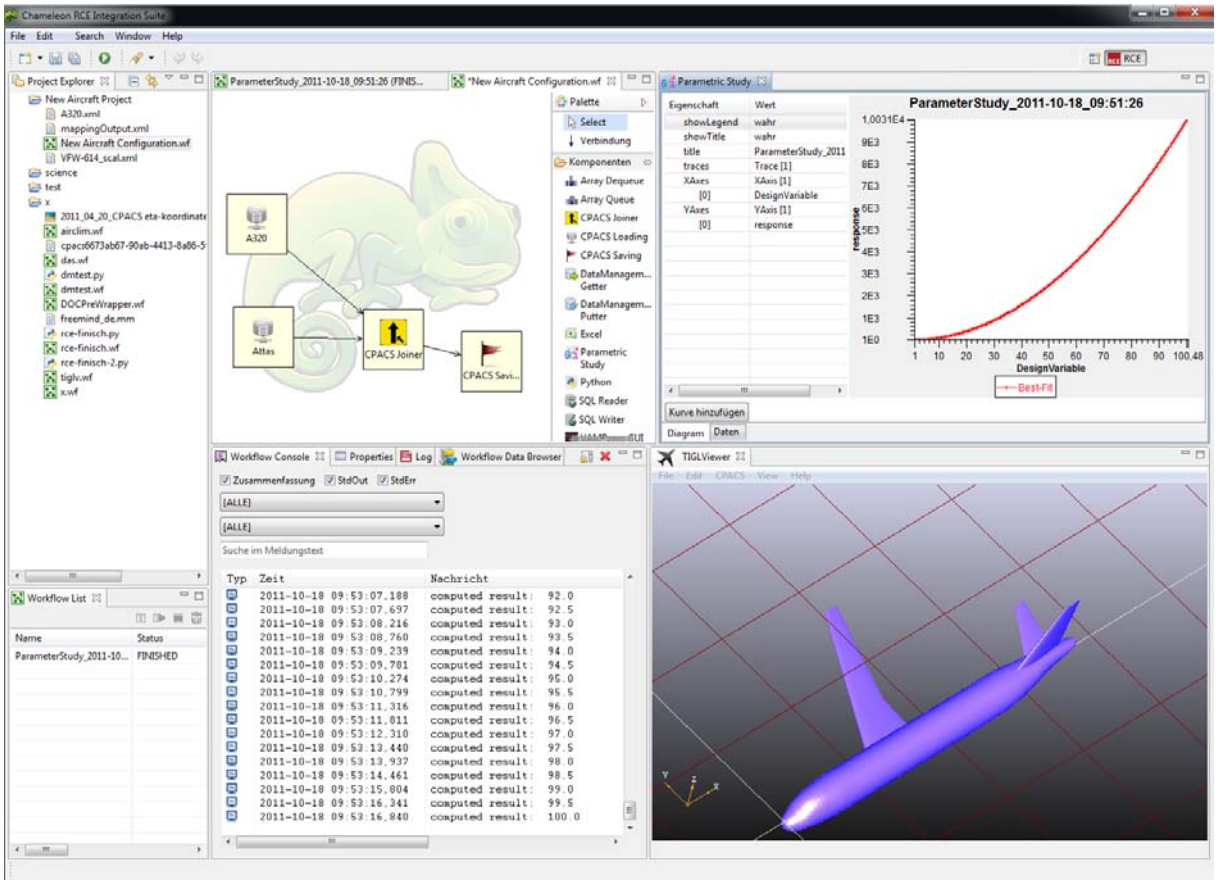

Figure 5. Chameleon based on RCE

demands of usability and quality were set in place. Out of these requirements the decision was made to take advantage of an already productive framework.

RCE was ready to use at that time and the software components it offers met the needs of Virtual Satellite. Both Virtual Satellite and RCE being based on Eclipse RCP and the Equinox framework allowed for an easy step by step integration. Thus, RCE was chosen as base framework for Virtual Satellite. The integration started on bundle level and has approached RCE's workflow engine now, which is in the process of integration. Figure 6 depicts the two main components Virtual Satellite is using. The privilege management as well as the workflow engine including its logging services.

Software Component Reuse and First Steps using RCE—The common base environment of Eclipse RCP allows for an easy integration of RCE without changing just one line of code in the Virtual Satellite. Aim of the first step was to enable developers to start using RCE features without being forced to change the current software design. Due to the component-based approach of Eclipse RCP this aim was achieved by copying all necessary bundles into the Virtual Satellite project and extending its configuration to start RCE services in the background.

As the first software component the logger system of RCE was reused in Virtual Satellite. All console logging messages were consequently replaced by adequate logger messages. Due to the ease of use this step was finished in just a few days and demonstrated a first successful use of RCE.

RCE's LDAP Authentication for CEF Studies—With the data model that is distributed among all engineers within the CEF study, data conflicts have to be avoided. The easiest way is by separating the work of the engineers, so that they not interfere each other. Following the hierarchical decomposition of the investigated system, only one engineer must be assigned to a *SystemComponent* at any time.

Together with RCE this was achieved by modelling *Disciplines*. These *Disciplines* consist of a name and only one engineer's user name who is responsible for it. Each *SystemComponent* is assigned to exactly one discipline, which is the only one allowed to alter data contained in it. RCE's LDAP authentication is now used to authenticate a user against the software. If an engineer is authenticating himself, the modeled disciplines will be checked for his user name. By this, the software analyses which disciplines have read and write permissions. Hence, the software decides to which *SystemComponent* the engineer has read and write permissions as well.

RCE's Excel Component for Individual Calculations—So far, the data models used in the CEF as well as in the CDF are based on ESA's IDM [19], [24]. As a consequence, most engineers supporting CEF sessions are used to it. Most of their work is based on years of preparation based and modeled in Excel. Accordingly, it is inevitable to support Excel in a new tool as well.

RCE provides an Excel wrapper. This wrapper is capable of writing and reading to and from Excel sheets. Additionally, it allows to directly embed Excel files into the developed

application and to interact with it in the known way. Virtual Satellite's editor has been extended to attach an Excel file with each individual *SystemComponent*. The file is attached and stored together with the data model. To directly connect it with the data model, the engineers can define inputs from the EMF model into Excel, run calculations and read the results back from the Excel file into the EMF data model. In this way engineers can continue to work with their known tool without missing the benefits of a distributed data model.

RCE's Workflow Engine Connected with EMF—The design of a new space craft is influenced by all disciplines. Strong dependencies exist among theses disciplines. Most of the disciplines use specialized tools as well to give the correct input to the overall design. Within aeronautics the approach of workflows is well established. It allows engineers to connect their tools sequentially together. Which as a result allows to give detailed answers taking the dependencies of the various disciplines into account. Within Virtual Satellite the challenge is taken to use the approach of workflows to aerospace. Even though, these workflows are not yet used in satellite design, they offer a new tool to the CEF to connect the various disciplines together.

RCE provides the workflow engine which already offers several workflow components that can be connected together. These components range from Python execution over Excel calculations to parametric studies. Within Virtual Satellite a new component was developed to connect the workflow to the EMF data model. Further workflow components to connect the workflow to domain specific tools, like orbit calculations or thermal analysis tools are considered but have not been approached yet. Nevertheless, with just the workflow components provided by RCE itself as well as the connection to the data model the CEF process has been enriched by an efficient tool. Since the design of a satellite is very complex engineers evaluate just a few design options in a CEF session. With RCE's workflow engine engineers can now start parametric studies on different parts of the design between the sessions to evaluate alternatives and to optimize the design without much extra software technology effort.

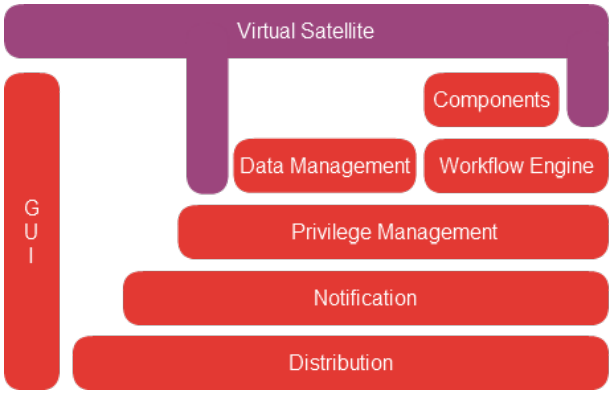

Figure 6. Virtual Satellite architecture on top of RCE

6. Conclusion

Summary

Out of the SESIS project, the DLR developed the open source software framework RCE (Remote Component Environment). It provides reusable base software components, like data management, workflow engine, or graphical user interfaces. RCE has been implemented in a component-based approach on top of Eclipse RCP. In conjunction with the current success of Eclipse in academia and industry, this offers a stable and modern basis for future developments. The software components RCE offer to developers have been implemented in accordance with their respective user requirements. These user requirements for the software components have been elevated over the past years of development and out of the various projects based on RCE.

Next to a shipyard application we build software from aeronautics and space on top of RCE. Presented examples are Chameleon, aircraft simulation environment and Virtual Satellite, software for design of satellites in an CEF (Concurrent Engineering Facility). Both of them reuse software components of RCE and implement specified ones to meet their individual requirements. They show that RCE helped to provide new important functionalities to their work environments with little effort and little changes to their internals.

Future Work

Chameleon supports multi-disciplinary preliminary aircraft design. Thereby, a key problem is optimization. Optimization is done per workflows including at least one optimizer workflow component and many tools computing different parts of an aircraft. Herby, on the one hand engineers need DLR optimizers specialized in DLR domains. On the other hand a set of basic optimizers for first quick evaluations are required. At the moment optimizers are only integrated individually for proprietary workflows in Chameleon. None of them are provided by RCE right now. Thereby, they are not reusable in other domain-specific software on top of RCE. As optimizers are more and more requested beyond Chameleon we will realize and provide a set of basic optimizers as RCE workflow components. Furthermore, we will integrate the DLR optimizer AutoOpti [25] and provide it as a workflow component. AutoOpti is a multi-objective optimizer for turbo machinery design. With them integrated RCE provides a reusable optimization environment for various use cases in aerospace and beyond.

Due to the amount of involved disciplines creating a simulation workflow in the multi-disciplinary preliminary aircraft phase is a scientific challenge. Thus, a created workflow in Chameleon contains many knowledge from different disciplines. But this knowledge only exists implicitly and will get lost as soon as scientists change. To avoid this knowledge leak Chameleon needs an integrated knowledge management. Because this requirement is a common one for aerospace software, we decided to extend RCE instead of Chameleon. So, we will integrate XPS into RCE, an expert system developed at DLR as well [4].

As described in Section 2 during research activities data is generated mostly all of the time. Each data has a history, so called provenance [26]. Data provenance contains a large amount of information. It gets more and more important in aerospace, e.g., at workflow execution in preliminary aircraft design. Hence, the aim is to enable RCE to gather provenance data and deduce information out of it. Therefore, we will provide common software components in RCE.

References

[1] O. Krämer-Fuhrmann, "RCE and SESIS - Service-oriented integration environment for collaborative engineering," *ERCIM News*, vol. 70, pp. 30–31, 2007. [Online]. Available: http://ercim-

news.ercim.eu/content/view/212/371/

[2] T. Forkert, H.-P. Kersken, A. Schreiber, M. Strietzel, and K. Wolf, "The distributed engineering framework TENT," in *Vector and Parallel Processing – VECPAR 2000: 4th International Conference*, ser. Lecture Notes in Computer Science, J. H. V. Palma, J.M.L.M.; Dongarra, Ed., vol. 1981. Berlin / Heidelberg: Springer, 2001, pp. 38–46.

[3] German Aerospace Center (DLR), "Remote Component Environment (RCE)." [Online]. Available: https://sourceforge.net/projects/rcenvironment/

[4] A. Schreiber, J. Rühmkorf, and D. Seider, "Scientific data and knowledge management in aerospace engineering," in *The Third International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP09)*. IEEE Computer Society, 2009, pp. 111–116.

[5] B. Venners, "Erich gamma on flexibility and reuse," Interview Transcript on www.artima.com, May 2005.

[6] D. Riehle, "Framework design: A role modeling approach," Ph.D. dissertation, ETH Zürich, Zürich, Switzerland, 2000.

[7] H. Mili, F. Mili, and A. Mili, "Reusing software: issues and research directions," *Software Engineering, IEEE Transactions on*, vol. 21, no. 6, pp. 528 –562, jun 1995.

[8] *Reuse Readiness Levels (RRLs)*, NASA Earth Science Data Systems - Software Reuse Working Group, April 2010.

[9] C. Liersch and M. Hepperle, "A unified approach for multidisciplinary preliminary aircraft design," in *CEAS 2009 European Air and Space Conference*, Manchester, UK, 10 2009.

[10] J. Preece, H. Sharp, and Y. Rogers, *Interaction Design - Beyond Human Computer Interaction*, 3rd ed. John Wiley & Sons Ltd, 2011.

[11] J. McAffer and J.-M. Lemieux, *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java(TM) Applications*. Addison-Wesley Professional, 2005.

[12] OSGi Alliance, "OSGi The dynamic module system for java." [Online]. Available: https://www.osgi.org/

[13] J. Wesselius, "The bazaar inside the cathedral: Business models for internal markets," *Software, IEEE*, vol. 25, no. 3, pp. 60 –66, 2008.

[14] Phoenix-Integration, "Modelcenter." [Online]. Available: http://www.phoenix-int.com/software/phx_modelcenter_10.php

[15] M. Litz, D. Seider, T. Otten, and M. Kunde, "Integration framework for preliminary design toolchains." in *CEAS Aeronautical Journal. Deutscher Luft- und Raumfahrtkongress 2011*, Bremen, DE, 09 2011.

[16] D. Böhnke, M. Litz, and S. Rudolph, "Evaluation of modeling languages for preliminary airplane design in multidisciplinary design environments." in *CEAS Aeronautical Journal. Deutscher Luft- und Raumfahrtkongress 2010*, Hamburg, DE, 08 2010.

[17] A. Bachmann, M. Kunde, M. Litz, A. Schreiber, and L. Bertsch, "Automation of aircraft pre-design using a versatile data transfer and storage format in a distributed computing environment." in *Third International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2009)*, Malta, GR, 10 2009.

[18] H. Schumann, A. Braukhane, J. T. Grundmann, R. Hempel, B. Kazeminejad, O. Romberg, and M. Sippel, "Overview of the new concurrent engineering facility at DLR," in *Proceedings of the 3rd International Workshop on System and Concurrent Engineering*, 2008.

[19] M. Bandecchi, B. Melton, B. Gardini, and F. Ongaro, "The ESA/ESTEC concurrent design facility," in *Proceedings of European Systems Engineering Conference EuSEC*, 2000.

[20] V. Schaus, P. M. Fischer, D. Lüdtke, A. Braukhane, O. Romberg, and A. Gerndt, "Concurrent engineering software development at german aerospace center - status and outlook," in *Proceedings of the 4th International Workshop on System and Concurrent Engineering for Space Applications*. European Space Agency (ESA), 2010.

[21] P. M. Fischer, V. Schaus, and A. Gerndt, "Design model data exchange between concurrent engineering facilities by means of model transformations," in *Proceedings of 13th NASA-ESA Workshop on Product Data Exchange 2011 PDE2011*, 2011.

[22] O. Romberg, A. Braukhane, and H. Schumann, "Status of the concurrent engineering facility at DLR bremen," in *German Aerospace Congress*, September 2008.

[23] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF Eclipse Modeling Framework*, 2nd ed. Addison-Wesley, 2009.

[24] H. Schumann, H. Wendel, A. Braukhane, A. Berres, A. Gerndt, and A. Schreiber, "Concurrent systems engineering in aerospace: From excel-based to model driven design," in *Proceedings of the 8th Conference on Systems Engineering Research*, 2010.

[25] U. Siller, C. Vo, and E. Nicke, "Automated multidisciplinary optimization of a transonic axial compressor," in *47th AIAA Aerospace Sciences Meeting*, Orlando, Florida, 01 2009.

[26] L. Moreau, P. Groth, S. Miles, J. Vazquez-Salceda, J. Ibbotson, S. Jiang, S. Munroe, O. Rana, A. Schreiber, V. Tan, and L. Varga, "The provenance of electronic data," *Communications of the ACM*, vol. 51, no. 4, pp. 52–58, 2008. [Online]. Available: http://elib.dlr.de/53871

BIOGRAPHY

Doreen Seider is deputy head of the department for "Distributed Systems and Component Software" of the German Aerospace Center's (DLR) Simulation and Software Technology division. She received her master's degree in computer science from the University of Leipzig in Germany in 2007. Since 2009 she leads the development of the Open Source software framework RCE (Remote Component Environment). She became the head of the group "Distributed Software Systems" at DLR in 2010. From then on she focusses on distributed data, knowledge and workflow management and mobile app development. In 2010 she worked several month at the Argonne National Laboratory in Web services fields.

Philipp M. Fischer is employed as research scientist at the German Aerospace Center in the department "Software for Space Systems and Interactive Visualization" and is focusing on the activities of model based software and systems engineering. He received his Diploma in electrical and computer engineering from the Leibniz Universität Hannover in Germany in 2007. Meanwhile he spent one year of studies in Computer Science at the Swinburne University of Technology in Melbourne, Australia. From 2007 until 2009, he supported Toyota's Formula One activities within the Department of Simulation and Performance Analysis at the motor sports headquarters in Cologne, Germany.

Markus Litz works at the German Aerospace Center (DLR) in the division of Simulation and Software Technology with the focus on software integration for simulation tools for the virtual aircraft predesign. He received his diploma in applied computer science in 2006 at the South Westphalia University of Applied Sciences Iserlohn, Germany. In 2010 he worked several months at the NASA Ames Research Center on an open source Apache module for XML indexing.

Andreas Schreiber received a diploma in industrial mathematics from Technical University Clausthal. He worked at the German Army (Bundeswehr) and Argonne National Laboratory. Now he is head of the Department for Distributed Systems and Component Software of the German Aerospace Center's (DLR) Simulation and Software Technology division. His research fields include high performance computing, grid computing, cloud computing, social networks, modern software architectures, user interfaces, software engineering, and software quality assurance. He organizes events for Python in High Perfomance and Scientific Computing.

Andreas Gerndt is the head of the department "Software for Space Systems and Interactive Visualization" at the German Aerospace Center (DLR). He received his degree in computer science from Technical University, Darmstadt, Germany in 1993. In the position of a research scientist, he also worked at the Fraunhofer Institute for Computer Graphics (IGD) in Germany. Thereafter, he was a software engineer for different companies with focus on Software Engineering and Computer Graphics. In 1999 he continued his studies in Virtual Reality and Scientific Visualization at RWTH Aachen University, Germany, where he received his doctoral degree in computer science. After two years of interdisciplinary research activities as a postdoctoral fellow at the University of Louisiana, Lafayette, USA, he returned to Germany in 2008.