A dynamic data integration approach to build scientific workflow systems

Arne Bachmann, Markus Kunde, Markus Litz, Andreas Schreiber
German Aerospace Center
Simulation and Software Technology
{Arne.Bachmann, Markus.Kunde, Markus.Litz, Andreas.Schreiber}@dlr.de

Abstract

The need for collaboration between individual scientific fields increases with the wish for more global engineering optimizations and assessment requirements. Since areas of research become more and more fine-grained domains of there own, it is still very desirable to cooperate with other experts with more chance than ever to gain synergies when science is scattered as today. But this exchange of knowledge comes only into consideration if it can be used in a simple way with at most an moderate initial effort. To this end a framework is developed that lets scientists easily use knowledge of others without the need to understand their work and technology completely. Furthermore a generic common data format based on XML technology is developed for exchanging and storing data between different domain-specific applications. To support all implementers, a twofold abstraction layer was introduced to encapsulate their knowledge shielding it from the technical environment.

1 Introduction

In today's scientific world there are a huge number of highly specialized but incompatible data formats available that prohibit researchers of different scientific fields to easily cooperate. At German Aerospace Center (DLR) the need for a common data exchange format to ease the origination of projects in aircraft predesign was identified and worked on to enable information exchange and global problem solving by researchers crossing the department boundaries. In their own self-contained knowledge domains all institutes and departments at DLR developed highly optimized tools, but for more global optimization tasks institutes need to cooperate. While this is basically possible using custom data converters and batch-scripts for simple tool chains, this approach is not powerful and flexible enough for today's demands on collaborative projects.

The goal of this paper is to describe a feasible solution for scientific workflows that can be fit to solve problems in many scientific domains without dropping existing workflows by imposing too restrictive constraints. This paper describes the development from the original data exchange problem up to the current scientific workflow system and finishes with the presentation of future plans.

The paper is organized as follows: Section 2 gives a deeper motivation for the work carried out. Section 3 gives an in-depth overview of the data format. In sections 4 and 5 the path from data integration to a workflow system is lined out. In sections 6 connected projects that use our approach are presented and sections 7 and 8 draw a conclusion including a brief evaluation of current outcomes and expected future extensions.

2 Motivation

For a successful cooperation in computer-aided optimization processes and evaluations, scientists and engineers need proper data interfaces to communicate knowledge and aggregated data. To do so, usually computer scientists help in designing data structures and software architecture as well as computer infrastructure to create systems for researchers that ease their work.

In each research area there are very good tools for specific problem areas available, often more elaborated and accurate than found in commercially available all-in-one solutions. The problem with dynamic creation of interdepartment cooperations was well known at DLR and let to the inception of the *TIVA* project in 2003 *(Technology Integration for the Virtual Aircraft)*.

The work carried out in the *TIVA* and *TIVA-2* projects started from the observation, that there is no suitable common data exchange format to describe aircraft configurations. While the standardization process is not yet finished and may continue while there are still new application areas coming into use, the combination and connection of different tools via this new data format became more and more important, as described in the following sections.

3 Common data format and data exchange

Despite the fact that there are a lot of data storage and transfer formats around in the scientific world (cf. CDF [1], HDF [2], Step [4], XSIL/BFD [12]), there was communion over the idea of creating a DLR-wide extensible standard format, originally designed for aircraft predesign.

3.1 Centralized dataset

Data exchange between different technical disciplines is very important for automated toolchains. This can be achieved by a common central database that is accessed by all applications for data loading and data sharing. This common database was formerly just one data set for one aircraft configuration, stored in a single XML file, thus bearing the general XML advantages of providing a well-known syntax and structure to wrap data [9]. To describe one aircraft configuration, the XML-based *Common Parametric Aircraft Configuration Standard* CPACS [10] was developed. The central data set contains one master file referenced from every application. Additional files in various formats can be linked to the main file via file paths. The ability to transparently create in-document references is also being worked on.

A file written in CPACS format describes an airplane configuration with all necessary data for the initial predesign phase. The CPACS format contains global data for describing an airplane as well as geometries and abstract data like cost factors or flight missions. The configuration is defined by larger hierarchically ordered construction units, e. g. wings and fuselages. The standard also defines how to store missions or how the simulation tools can write their results back to the central data set and into satellite files.

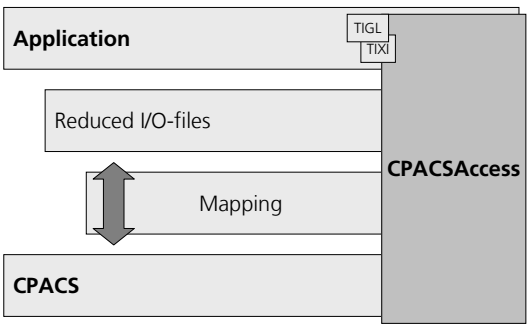

Figure 1. Overview of the data flow between the central data set and the applications.

3.2 Supporting Tools and Software

A software suite was developed to enable connected simulation software to communicate with each other via a central data set in CPACS format. CPACS is completely based on an XML-schema [15] that describes the structure and valid data containing aircraft configurations. To enable standalone applications to access XML formatted files there are two different approaches:
— Either the application reads and writes XML data by itself,
— or a filter program must be written that transforms the incoming XML data to the native format of the application as input, and the application's result is transformed by the same or another converter back to CPACS-conform XML. This is especially useful when the software is known to work for a long time, cannot be changed because of license reasons (is commercial and/or not open source), or there is no expert for the source in the project any more. This approach shields the application developer completely from dealing with XML, because the converter programs could be written independently from XML experts only with knowledge of the native file format of the application. As shown in Figure 1 there are two helper libraries for data manipulation, namely *TIXI* and *TIGL*.

3.2.1 TIVA XML Interface (TIXI)

Many applications use input files based on quite simple data structures. Often single floating point or integer numbers are used, and their meaning depends on the exact position of these numbers in the input or output file. More advanced types of these files are name/value pairs or lists of numbers to reflect vector or array data. Based on these requirements it was decided to design an API that supports the developer with access and archiving their data via a simple interface to XML. On top of the full-fledged XML-library libXML2 [5], the easy-to-use C-library *TIXI* was designed to shield the developer from the complexity of libXML2 and XML processing in general when performing simple tasks like writing an integer or reading a matrix. In most cases only the XPath and/or the corresponding value is required to read or write data. Language interfaces for Fortran and Python are also provided.

3.2.2 TIVA Geometry Library (TIGL)

Another library we developed is for easy processing of geometric data stored inside CPACS data sets, and is called *TIVA geometry library*. With *TIGL* it is possible to directly execute geometric functions on fuselage and wing geometries, with plans to add functions for other aircraft parts (e. g. engines).

TIGL is written in C++ and uses the *TIXI* library to access CPACS data sets, while leveraging data manipulation of all supported geometry types to e. g. build up a 3D-Model of the contained aircraft in memory.

At the time being only wings and fuselages are supported, with more to come. The functional range of the library goes from the computation of surface points in Cartesian coordinates up to export of airplane geometry in different file formats (e. g. IGES [3] and STL [7]).

Beside these computational functions, *TIGL* could be used to obtain information about the geometric structure itself, for instance how many wings and fuselages the current airplane configuration has. The library provides external interfaces to C and Fortran and could be used to build up new supporting programs or to extend existing applications with *TIGL*-enabled functions.

3.2.3 CPACSAccess

To get the necessary data out of the central data set and for storing application results back to it, a converter called *CPACSAccess* was developed to manage this data exchange. It is worth noting that import and export of data is controlled via mapping files that build the second abstraction layer around each tool: *CPACSAccess* generates the XML input file with data from the central data set via execution of rules specified in a mapping file. Thus the complete data set is transformed in a stripped and more compact XML input dataset, enabling existing applications to use their custom XML format to continue being used. After running, applications have to write their result files back in any XML schema they choose, which is then taken by *CPACSAccess* again and merged into the central data set according to the rules contained in the (output-) mapping file (see Figure 1).

4 Data and application integration

The collaboration projects were originally undertaken via simple batch scripts calling tools and larger applications in a predefined manner, thus leveraging repeated execution of process chains. Naturally after some time the approach was no longer sufficient when requirements became harder with regards to flexibility of process definition and custom user manipulation of the tool chains as well as data visualization. This called for a data and software integration framework which was finally decided about at the end of the *TIVA* project.

4.1 Framework selection

The choice of a framework was made in 2006. An analysis [11] collected relevant requirements, defined evaluation criteria and formulated test cases. As experience in the field of this type of frameworks was already available the analysis consists only of requirements specialized for projects at DLR and their classification. A market analysis inspected several frameworks, hence a pre-selection of six frameworks was decided about. These candidates were tested by individual groups for the test cases defined before. The results were recorded in a result-matrix.
In the following an extract of the requirements and the results of the framework selection is given.

The functional requirements can be divided into five different categories that build the fundament for the defined test cases:

- *Buildup and functionality of process chains.* Several functional requirements which are essential for the functionalities of the framework regarding a process chain (e. g. automatic execution of process chains, ability to store/reload them)

- *Usability.* The usability requirements regarding the user-group (e. g. graphical user interface, parameters should not be limited)

- *Additional required functionality.* Essential requirements that have no direct connection to process chains (e. g. support of scripting languages, availability of optimization algorithms)

- *Price and legal issues.* E. g. floating user license or fixed desktop licenses

- *Documentation and support.* Willingness of the providing company to put effort into support

The discovered test cases were evaluated within defined evaluation criteria ranging in six levels from *very unsatisfactory* up to *very satisfactory* .

Besides the functional requirements all technical needs were to be considered. For this an information matrix was assembled that shows the availability of each criterion for all frameworks examined. As the final result the *ModelCenter* framework from Phoenix Integration [6], version 6.0 at that time, was the most appropriate framework.

4.2 Integration of the Central Data Base into the Framework

In a first version, a Java plugin for *ModelCenter* was developed to manage XML storage in a central location. Each application attained access to the same, global data set and was called in a fixed order. XML processing was undertaken by using the Apache Xalan/Xerces libraries [8].

4.3 Wrapping the applications

The main advantage of the concept described in this paper is a huge flexibility regarding the integration of an application. The interface between the framework and the requested application consists of several abstraction layers and a separate configuration file. The concept itself is hereby called 'tool wrapping'. Wrapping a tool or application is hereby a piece of software which encapsulates the base application and hides it from the environment outside of the tool wrapper. The primary advantage is to have an abstraction layer for the outside environment which again can use the interface provided by the tool wrapper. In this implementation type the tool wrapper needs to be customized for the wrapped application. The approach described at this point implements a two-sided abstraction layer. On one hand there is the usual 'environment-to-toolwrapper' abstraction, on the other hand the tool wrapper is implementing a generic interface to the wrapped application. This is done by a specific configuration approach, described below. This two-side concept creates the positive situation, that an already existing application does not need to be customized much. Figure 2 shows the hierarchy of these layers and the general sequence of processing. The CPACS data format represents the whole set of all exchange data and builds the base for all operations. The processing inside the tool wrappers can be described most suitably in a chronological manner. Following is an abstract description of all steps from the application's start in the process chain to the end of execution:

1. *Reading of the input mapping file.* For each tool there is a mapping file stored on its server, containing an arbitrary number of tool-specific transformation rules. Each mapping rule will be performed on the source CPACS data. The result is a new generated input data file which is usually a subset of the CPACS data.

2. *The wrapped program starts.* The application wrapped by the toolwrapper will be executed. It should read the input-data-file created by the mapping file. The result of the execution of the application will be written to an output data file.

3. *Reading of the output mapping file.* Each mapping rule will be performed on the output data and merged into the original CPACS file. The result is an updated CPACS data set.

A mapping definition file is a set of rules that is processed on an XML data document to transform it into a new document, not unlike XSLT stylesheet processing. The main features used in this concept are source/target pairs and loops as well as other options regarding the XSLT technology. In addition several *modes* are enclosed. On the one hand an *append* mode that adds a further sibling element to already existing elements in the data. On the other hand the *delete* mode that replaces an existing element by the new one; other useful modes are conceivable and in progress.

Below there is a simplified example of CPACS data, a mapping file and the transferred result-file.

```
<configuration>
  <global>
    <pax>3</pax>
    <span>5</span>
    <range>7</range>
  </global>
</configuration>
```

Listing 1. Reduced CPACS data

```
<map:mapping>
  <map:source>
        /configuration/global/pax
  </map:source>
  <map:target>
        /toolInput/data/var1
  </map:target>
</map:mapping>
<map:mapping>
  <map:source>
        /configuration/global/span
  </map:source>
  <map:target>
        /toolInput/data/var2
  </map:target>
</map:mapping>
```

Listing 2. Reduced Mapping file

```
<toolInput>
  <data>
    <var1>3</var1>
    <var2>5</var2>
  </data>
</toolInput>
```

Listing 3. Transferred result-file

The second abstraction layer is based on a central configuration repository, realized as an XML file. The tool wrapper is pre-configured by a client-side GUI with three input parameters 'server', 'tool' and 'version', and fetches its actual tool configuration from the central configuration repository as of these values. The tool configuration can be addressed in different ways, e. g. its local configuration repository can be specified by an environment variable or put into any of several of standard path locations to be discovered automatically. The main reason for a globally available configuration is a central administration and the traceability of a single configuration as well as a way for process chains to always use the same configuration.

As often in systems providing integration capabilities in our project there was the requirement for the infrastructure to spread over heterogenous computing systems, including different operating systems, hardware architectures and a variety of software configurations (for example software

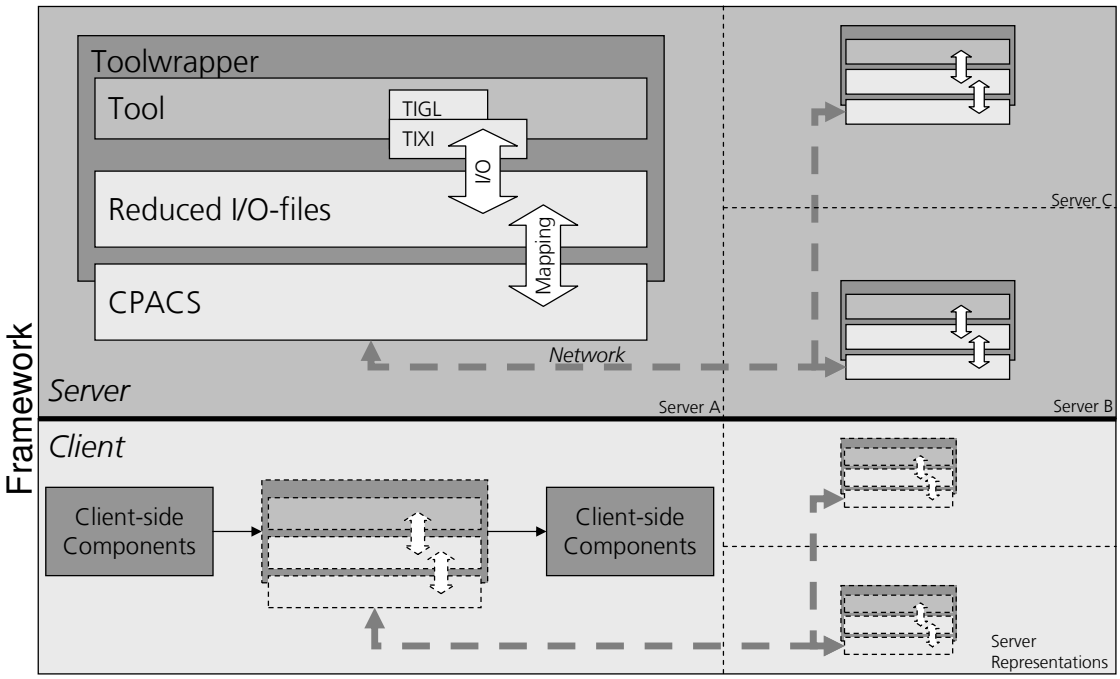

Figure 2. Overview of the data flow and the general architecture of the framework.

distributions and network/firewall configurations). Most of these points were already settled by using *ModelCenter*.

5 Towards a scientific workflow system

Up to this point we came from a data integration approach to using an application integration framework. Multiple components of wrapped applications and tools could be connected in *ModelCenter* and executed via standard backward scheduling or custom script schedulers that defined the process chain. At the time being it has come clear that the approach with a central data set carrying the initial data, all intermediate and the final computing results is no longer a feasible option.

It became clear that a real workflow philosophy would be much more appropriate for defining data flows and successive manipulations of the data transferred between the components. In this paper, we refer to this by *Scientific workflow system with dynamic data integration*, as explained in the following part.

5.1 Scientific workflow system

A scientific workflow differs from a ”standard” business workflow in several details: Scientific processes are carried out on a trial-and-error basis and are often only partially defined upon conception. They are build for knowledge gain and reuse [16]. The main qualities of scientific workflow (management) systems that we are interested in can be summarized as follows:

- To enable collaboration and aggregation of large scientific applications over locally and architecturally distributed systems via client/server architecture
- To enable some kind of basic provenance of workflow processes for a larger number of people
- To increase the flexibility and availability of tools and information in cross-department projects
- To create a component library that allows easier startups of coming projects
- To unify data formats and help avoiding modal fragmentation
- Authentication, security and auditing

To enhance the chosen standard integration framework by simple workflow management tasks and abilities, we created several Java plugins and data handling routines to work around the frameworks limitations.

5.2 Dynamics of data connections

One special requirement of our approach seldomly found in other workflow systems is the ability of software compo-

nents in the workflow to arbitrarily change the number of variable connections to other components based on computed decisions during runtime. We had to implement this feature on top of *ModelCenter*, since as an requirement we had no fixed number of variables and component variable links that would have had to be connected by the user *a priori* (before the process is started). There are at least two reasons for this requirement:

- One component and therefore one wrapped scientific application may have different working modes, determined by an input variable. This way two very different output data sets may be created, based on a runtime decision, that would need be connected completely different in a statically connected integration system, thus impeding our approach. To always connect all variables for both modes would blur the actual meaning of a data connection and would pose hard-to-find errors.
- The second reason for the necessity of a variable number of connecting variables – elsewhere defined as *channels* [14] – is for instance an array of simple data types or arrays of complex and again variable data structures. One viable solution for static variable connection systems would be to always connect all potential interesting variables, thus bloating the process chain (e. g. by always providing 100 connection "slots" for changing number of variables).

The solution presented here is by means of XML files: Each component offers only one string variable for input and output transfer. This way an updated XML tree can be send to the following components in the workflow. The workflow designer doesn't have to connect all variables one by one as usual in *ModelCenter*, but simply connects one complete XML string variable and is settled, the hard tasks being performed by the software infrastructure.

To achieve this goal, the logic of the plugins we developed is as much framework-independent as possible. Main tasks like input and output-mapping are done without the use of framework functionality. Furthermore is it essential that the framework knows about the data and changes made to the data to resolve necessary computation loops inside the workflow.

6 Examples from DLR projects

This section provides a glance at the projects using the technology described in this paper. Even though each project has different goals, the technology behind the scenes is the same. The idea in these projects is to perform predesign in the aviation domain, but their approach is not always congruent with each other. However, there are even more projects planned in the future to implement this concept.

TIVA – *Technology integration for the virtual aircraft.* The main goal of this project is to perform technology assessment regarding collaborative predesign of civil aircrafts. A proof for a working recomputation of an aircraft configuration was mandatory in this project. The first revision of the central data set structure that now is the CPACS was made, and the desired framework for software integration *(ModelCenter)* was chosen.

UCAV-2010 – *Unmanned combat air vehicle.* This project simulates the development of a military aircraft optimized for stealth capabilities. Existing knowledge gained in the *TIVA* project is taken into account. In this project an adaption regarding the central data set, new software libraries and an expert system were added.

EVITA – *Evaluation of innovative turbine engines.* The technology and data used in this project is quite similar to *TIVA*, but is much more focused on the experimental predesign of engines than on the whole aircraft, thus shifting the amount of data for turbines to a more detailed level while compressing data for the aircraft to only a few basic values. Therefore, the focus is set on other details, but *TIVA* as well as *EVITA* can benefit from each other in a technological way.

CATS – *Climate optimized air transport system.* The goal of this project is to optimize aircrafts and aircraft missions regarding their climatic effects. In this project the technology described in this paper is used in combination with another tool developed for climate assertions.

Although these projects differ in their domains strongly from each other, however they use the same technology for data and software integration into the framework. Besides similar technologies used, the generality of our approach can be shown by the *CATS* project, in which a wholly new facet is included by optimizing an aircraft to a non-predesign oriented technical aspect like climate impact. The benefit by using an flexible data format like CPACS as an internal standard at DLR becomes clear when considering that in the future the results of all projects can be used within future projects with very low effort in integrating different technology and knowledge domains. All projects presented here might have positive effects on current and future projects. With this solution it is therefore not only possible to share knowledge on a company-wide basis, but also to encompass different fields in interdisciplinary projects.

7 Current and future work

Currently our concept is implemented and working as described in this paper. However, there are a few points which are planned to be improved:

At the moment the CPACS' structure is being augmented to have a more generic data format to address a diversified intended audience. The ideal is to have a data format which can be used company-wide or at minimum across similar-aligned projects where data exchange is desired or necessary. Therefore a data structure has to be developed where all possible use cases should be considered and still leave the structure as simple and generic as possible. A first meeting with all current and future project partners took place at the end of 2008. The main priority of an adjustment of CPACS is to let it become usable by all partners and to keep its functionality for data exchange in and across projects.

Besides the improvement of the data structure a workflow description standard for storing and processing workflow information will be decided upon. It is imaginable to use an already existing standard like BPML *(Business Process Modeling Language)* or to create a more appropriate one. Additionally the existing integration framework *ModelCenter* will be phased out during the next project phases and replaced by the *Reconfigurable Computing Environment* RCE [14], but also integration into other frameworks is possible. This again shows the versatility of our approach and the advantage of this concept with the possibility of easy adaption to other environments.

Here again the advantage is that RCE is an in-house product where sufficient developer knowledge is available and which can be customized close to the requirements requested by project partners. In addition, RCE has just been released as an open-source software. This means that interested parties can be involved into the work presented here very quickly without a high initial expense regarding licensing and pricing found in framework vendors.

Another interesting point yet to be discussed is data management and storage. Currently CPACS is used as a data storage container format as well as a data transfer format. Future analyses could work on integrating CPACS into an XML database or on optimizing the runtime data streams. Here the simplest approach might be to compress the transfer data by standard compressing algorithms. The second solution would be to develop a binary version of CPACS, especially designed for quick and robust data transfer between components.

Data management as of the "management" part in (scientific) workflow managements systems is one area currently being put a lot of energy in. At DLR there are metadata solutions for scientific data management available, too [13]. We are exploring the term of management a step further by managing anonymous data of different servers that run components at run time, because the amounts of data being transferred is foreseeable to rise by several orders of magnitude in future projects, thus calling for a better storage solution.

8 Conclusion

The need for collaboration of individual scientific fields increases step by step by the ongoing consideration of an environment as a whole in a scientific view. As each part of this co-operative process produces state of the art knowledge but wants to use the knowledge of other domains as well, the need for an easy-to-use approach and a common data transfer and storage facility is high. Regarding this evolution the framework developed boosts its right to exist with every usage. In this paper a first insight into the possibilities of collaboration and knowledge exchange was given. The motivation and technical decisions were pointed out and the technology used was described. With respect to other projects and future developments within the domain of data integration into workflows, this paper presents a simple but powerful approach to dynamically connect data and information from different knowledge domains to a collaborating workflow system.

References

[1] Common data format. `http://cdf.gsfc.nasa.gov`. [Online; 2009-03-10].

[2] Hierarchical data format. `http://www.hdfgroup.org`. [Online; 2009-03-10].

[3] Initial graphics exchange specification. National Bureau of Standards.

[4] Iso 10303-1: Industrial automation systems and integration–product data representation- and exchange–part 1: Overview and fundamental principles. I. TC 184/SC4 ISO Standard. 1994.

[5] libxml2. `http://www.xmlsoft.org`. [Online; 2009-03-10].

[6] Process integration & design optimization. `http://www.acel.co.uk/pdfs/designsimulation/`. [Online; 2009-03-19].

[7] Stl file format. 3D Systems. `http://www.ennex.com/~fabbers/StL.asp`. [Online; 2009-03-10].

[8] Xalan and xerces. `http://xalan.apache.org`. [Online; 2009-03-10].

[9] Extensible markup language (xml). `http://www.w3.org/XML/`, 2008. [Online 2009-03-10].

[10] DLR. Common parametric aircraft configuration standard.

[11] M. Hepperle and J. Agte. Framework test. unpublished.

[12] J. Myers and A. Chappell. Binary format description language (bfd). `http://collaboratory.emsl.pnl.gov/sam/bfd/`. [Online; 2009-03-10].

[13] T. Schlauch and A. Schreiber. Datafinder. a scientific data management solution. In *PV 2007*, 10 2007.

[14] D. Seider. Rce homepage, 2008. [Online; 2009-01-09].

[15] E. van der Vlist. *XML Schema: The W3C's Object-Oriented Descriptions for XML*. O'Reilly, illustrated edition, 2002.

[16] M. Weske, G. Vossen, and C. B. Medeiros. Scientific workflow management: Wasa architecture and applications. Technical report, University of Mnster and University of Campinas, January 1996.