An EMF-based Toolkit for Creation of Domain-specific Data Services

Andreas Bender, Stefan Bozic and Ivan Kondov
Steinbuch Centre for Computing (SCC), Karlsruhe Institute of Technology (KIT),
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Karlsruher, Germany

Keywords: Metamodel, Eclipse Modeling Framework, Dataflow, Data Model, Workflow, Application Integration, Web Service.

Abstract: Development of composite workflow applications in science and engineering is troublesome and costly due to high heterogeneity of data representations and data access interfaces of the underlying individual components. As an effective solution we present a generic toolkit enabling domain experts to develop data models and automatically generate a self-contained data access service. We defined a custom metamodel based on Ecore which can be readily used to create domain-specific data models. Using the generated data access service, instances of the modeled data residing on heterogeneous and distributed resources, such as databases and cloud data stores, are accessible from the individual application components via a language-independent Web service interface. We discuss the framework architecture, the toolkit implementation, the deployment process, as well as the performance of the data access service. Workflow designers as target users would benefit from the toolkit by using it for rapid and cost-efficient application integration.

1 INTRODUCTION

The complexity of applications for simulation and data analysis in science and engineering has increased dramatically over the recent years. Such applications, often designed in the form of generic workflows, combine multiple software components originating from diverse application domains. Thus, developing such a composite application requires significant effort either for coordination of experts from these domains or for learning multiple domain-specific program codes by individual researchers. Scientists designing workflow applications face a rapidly increasing number of different programs which carry out very often the same functions. Thus the time necessary to implement changes, which are typically made on a short notice and rather frequently, has increased substantially. On the other hand, current environments for simulation and data analysis provided at computing centers are still too complicated for use by non-experts. Since computer simulations and data analyses are typically planned and carried out by scientists and engineers who are often non-experts in the technical field of computer science and software engineering, a corresponding easy-to-handle software infrastructure has to be provided. For this purpose, the paradigms of service-oriented architecture (SOA) (Erl, 2005) and model-driven engineering (MDE) (Schmidt, 2006) have been adopted to develop systems at such a level of technical abstraction that application domain experts can develop composite applications by integrating existing components following their design rather than spending efforts with the technicalities of the underlying computing environment. For instance, the SOA concept has been recently adopted to integrate multiple components into workflow applications for multiscale materials simulation (Kondov et al., 2011; Bozic et al., 2012) motivated by demands in the community.

The major challenge we met with constructing a workflow application is the handling of data exchange between the workflow steps which is often referred to as *dataflow*. Data modeling in workflow's specification has been shown to be very important to avoid potential data flow problems (Sadiq et al., 2004). Although complex data must be stored in such a way that all workflow steps can easily access it, many program codes usually cannot use the same data source in practice because they have mutually incompatible data representations, heterogeneous data formats or non-uniform data access interfaces. A common strategy employed in multiple application domains bases on format converters between the workflow steps which operate for a limited set of supported formats. This strategy was found unsatisfactory because it requires

Bender A., Bozic S. and Kondov I..
An EMF-based Toolkit for Creation of Domain-specific Data Services.
DOI: 10.5220/0004701900300040
In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2014), pages 30-40
ISBN: 978-989-758-007-9

tedious, frequent and error-prone reimplementation of converters for each combination of simulation or analysis codes in a workflow application (Bozic and Kondov, 2012). Moreover, multiple conversions of large data can become a bottleneck and even make the workflow simulation unfeasible. Also, the converter solution is usually restricted to storage on a file system which introduces the need to manage additional metadata along the workflow.

To treat the dataflow more efficiently different domain-specific non-transferable solutions have been developed in diverse application domains, e.g. in fusion plasma engineering (Manduchi et al., 2008), nuclear magnetic resonance (Vranken et al., 2005; Fogh et al., 2005; Nowling et al., 2011), computational chemistry (Murray-Rust et al., 2011; Birkenheuer et al., 2012), molecular engineering (Dubitzky et al., 2004; Sild et al., 2005; Sild et al., 2006), oil and gas industry (Rahon et al., 2012) and materials science (Bozic et al., 2012; Bozic and Kondov, 2012), whereby some of them have employed a model-driven approach. For example, the integration of COSMOS, a program code applied in the computational nuclear magnetic resonance, into the CCPN data model was straightforward (Schneider et al., 2012) while our attempt to adopt the same data model in other domains, e.g. in computational materials science, was less successful. In particular, we realized that a generic data model (a metamodel) and a generic tool is necessary to allow any domain expert in the role of workflow designer to develop domain-specific data models. Moreover, the data should be made accessible from each individual application code via a language-independent interface, e.g. a Web service, for rapid implementation of pre- and post-processors and construction of workflows from standard components.

In this paper we present a generic solution for modeling and management of data in scientific workflows in a simple and uniform way. We report on a development of the concept from (Bozic and Kondov, 2012) resulting in a novel SOA- and MDE-based framework using modern standards for Web services. We will describe the implementation of the framework and discuss its functionality and general applicability. This service-oriented framework is based on a custom metamodel and domain models from which a complete service can be created with the Eclipse Modeling Framework (EMF).

Elsewhere (Bender et al., 2013) we have demonstrated the applicability and the practical benefits of the framework in the use case of a composite multiscale modeling application in computational science. Here, we will focus on the program architecture and report on the technical implementation of the framework as a toolkit for model-driven automatic generation of data access services. The paper is organized as follows. In the next Section 2 we introduce the requirements and then in Section 3 discuss the conceptual design and program architecture of the toolkit. In Section 4 we discuss our specific selection of technologies which were used for the toolkit implementation and the generation process as described in Section 5. Further, in Section 6, we outline the unique advantages of our approach and analyze the performance of the data access service generated for a specific use case. In Section 7 we review related approaches in the context of the present work. Finally, in Section 8, we summarize the key results of this work and suggest directions for future work.

2 REQUIREMENTS ANALYSIS

In order to develop and implement a concept for the framework we have considered all lessons learned from previous experience and the requirements of data modeling in different domains. In the following we will outline the essential requirements. The framework should

- be generic and domain-independent so that it can be used in different domains with no further modification.
- act as a bridge for access to heterogeneous and distributed storage from distributed workflow applications thus providing two interfaces — one for the storage and one for the application side.
- provide a modeling environment, containing a graphical editor, for creation of data models for domain-specific needs.
- provide utilities for fully automatic code generation of all components because it will only in this case provide a low-effort and low-threshold solution for the end-user.
- provide a language-independent application access interface, e.g. a Web service.
- provide an abstract storage access interface allowing applications to connect to distributed and heterogeneous data storage resources e.g. relational databases, simple files or cloud storage services such as Amazon S3.

The implementation of the framework will be a toolkit providing interfaces that assist the user in all steps of the service creation: from the construction of a data model, through code generation of a service, up to the deployment as data access server. The toolkit should be operating system-independent and

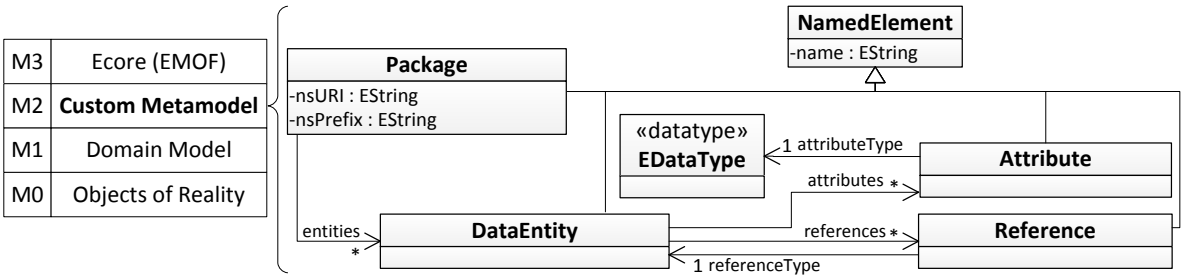

Figure 1: Custom M2-layer metamodel.

preferably based on open-source technology. In addition we aspire to use the modular service platform OSGi which enables reuse of components, reducing the complexity during developing components, versioning, simple deployment and dynamic updates.

3 FRAMEWORK ARCHITECTURE

In this section we describe the architecture of the framework starting with the selection of the metamodel from which all domain-specific data models will be instantiated. Afterwards the concept of a Service Generator which transforms a given data model into a set of kernel classes is explained. Finally we will have a closer look at the generic kernel which provides two interfaces connecting a storage resource with arbitrary workflow applications.

3.1 Metamodel

The creation of individual domain-specific models, as aimed by our framework, is part of an MDE process. The Meta Object Facility (MOF) is an MDE standard defined by the Object Management Group (OMG, http://www.omg.org/) forming the base for building custom metamodels as shown on the left hand side of Fig. 1. The M3 layer is the most generic one and defines a standard for the creation of meta-metamodels which are needed to build metamodels. Metamodels are resident in the M2 layer. The most prominent representative of this layer is the Unified Modeling Language (UML) (OMG, 2003). However, UML class diagrams contain methods or other components beyond those needed to model data entities and relationships. This UML complexity should not be exposed to the user constructing domain-specific models. To solve this problem we used the M3 (Ecore) layer to create a custom simple data-centric metamodel (in the M2 layer) as alternative to UML which can be understood by domain experts having no IT knowledge. This custom metamodel is restricted to the minimum necessary for modeling and managing data and forms the base for domain models which are part of the M1 layer. Thus, the proposed custom metamodel is innovative because it implements a practical trade-off between genericity and instant usability. A concrete instance of the M1 layer is a model that describes real objects and related data of the M0 layer. The objects of reality pertinent to the M0 layer represent domain-specific data units that users would like to make persistent in a storage instance. The meaning and value of such data units depend on the corresponding domain. For example, in computational materials science such objects could be atoms, molecules and chemical bonds between atoms. We designed the custom metamodel primarily for simulations and data analysis in scientific and engineering applications. However, owing to capabilities of Ecore the M2-layer metamodel can be readily extended to allow typing of data structures such as e.g. data cubes.

The diagram in Fig. 1 shows the newly defined metamodel which is based on Ecore. The main element of the metamodel is the *DataEntity* which is used to model concrete data entities. These entities contain *Attribute*s of different data types and may also contain *Reference*s to other entities. With these references relationships or dependencies between entities can be defined. DataEntities are combined in a *Package*. A package is necessary to define identification information, such as a URI, in order to distinguish between different data models. It is planned to extend the metamodel with the concept of inheritance allowing the creation of more complex data models with less effort. Also it is intended to further investigate the applicability of the metamodel in different more extensive domains.

3.2 Service Generator

Figure 2 depicts the generation process for a domain-specific Data Access Service. A domain expert uses a graphical editor to create a domain model based on the metamodel. Additionally the user has to define

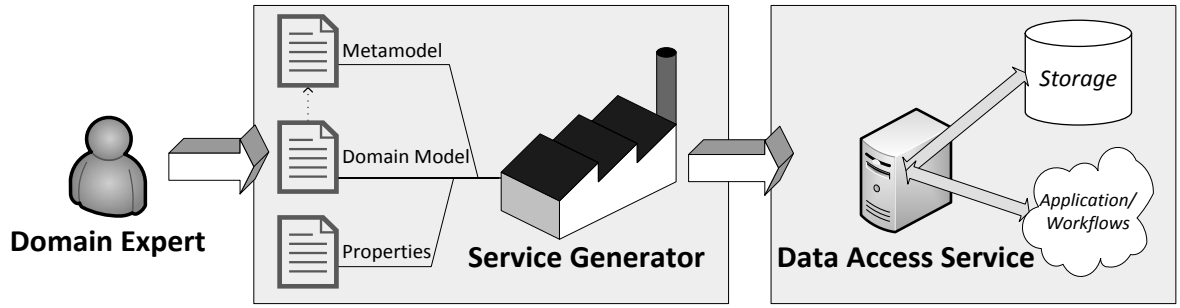

Figure 2: From a domain model to a Data Access Service.

a set of properties that are needed to gain access to the target storage resource, typically including a connection URL and security credentials. Another set of properties is mandatory to define the Web service interface. The metamodel, the domain model and the service properties are then used by the Service Generator to transform the model to the final Data Access Service.

3.3 Data Access Service

To support a large number of applications and make the Data Access Service accessible via the Internet we adopted the SOA paradigm making use of Web services technologies and open standards. The main component of the Data Access Service is a kernel that connects applications via a Web service with a data storage. The kernel has a three-layer architecture shown in Figure 3. The persistence layer handles the mapping between data objects and storage entities. A well known technology to realize this is O/R (object-relational) mapping (Ambler, 2012). Unfortunately this technology is limited to relational databases and we decided to use a storage-type independent solution for this part. The toolkit users should be able to choose a storage type for their data which fits best to their problem domain, for instance relational, graph-based, web-based or document-based data stores. The purpose of the representation layer is to marshal/unmarshal data objects to a transport format, e.g. XML or JSON. The most complex layer is the resource layer which defines the Web service interface. The layer acts as controller mapping the Web service operations to CRUD (Create, Read, Update, Delete) operations of the persistence layer. Furthermore it is responsible to deliver and receive marshaled data objects using the representation layer.

4 TECHNOLOGY SELECTION

Previously (Bender et al., 2013) we have briefly introduced the technology stack which has been derived from the requirements and the framework concept. In this section we will discuss the pros and cons of different technologies available for the implementation. We suppose that such a discussion is interesting in the domain of model-based engineering and can be used to improve existing modeling tools.

4.1 Model Development

The Eclipse Modeling Framework (EMF) has been established within the Eclipse Modeling Project (http://www.eclipse.org/modeling/) providing model-based development technologies and a large collection of modeling tools for the Java programming environment, including graphical tools for construction of models and metamodels, editors and for generation of source code. Since we have the requirement to define a metamodel and corresponding graphical editors as well as a code generator, therefore EMF seems to be the technology of choice.

4.2 Model Transformation

The Eclipse Model To Text (M2T, http://www.eclipse.org/modeling/m2t/) project enfolds three generator tools which are capable of transforming concrete models into text or source code using different template languages: Java Emitter Templates (JET), Xpand and Acceleo. JET uses a language that is similar to Java Server Pages (JSP) while Xpand uses a self-developed template language and the template language of Acceleo is an implementation of the MOF Model to Text Transformation Language standard (MOFM2T) of the OMG. Xpand and Acceleo assist the template developer with

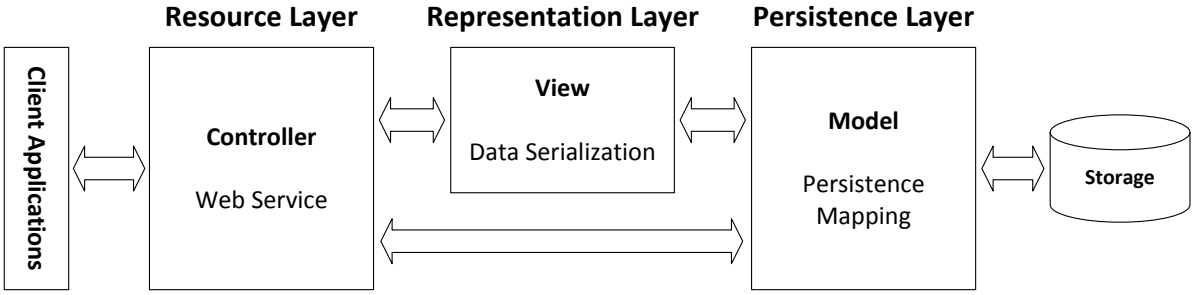

Figure 3: The Data Access Service encapsulates a kernel with a three-layer architecture.

rich text editors that provide syntax highlighting, syntax validation and code completion. The standard template editor of JET is yet not fully developed. The template editors of Xpand and Acceleo look very similar in respect of usability and function volume. Because, in addition, Acceleo follows the MOFM2T standard we decided to build our generator tool with the help of Acceleo.

4.3 Editor Development

The Graphical Editing Framework (GEF) supports developers in creating graphical editors for the Eclipse Platform. Such editors provide convenient means to create complex objects such as state diagrams and process flow editors. To simplify the creation of graphical editors based on EMF metamodels and GEF, the Graphical Modeling Project (GMP) provides generation components and runtime infrastructure. Especially the Graphical Modeling Framework (GMF) allows the development of diagram editors based on Ecore models without programming any line of code.

4.4 Persistence Layer

A main requirement of the service is the possibility to save data in different kinds of storage. A flexible way to handle this issue provides the standard Java Data Objects (JDO) (Oracle Corporation, 2013b). JDO is an annotation-driven framework which maps Java objects to storage entities. The reference implementation of JDO is DataNucleus (http://www.datanucleus.org) which supports a large set of various storage types such as RDBMS (Oracle, MySQL), map-based (HBase, Cassandra), document-based (MongoDB) and web-based (AmazonS3, Google Storage) storages.

Several EMF-based frameworks address the topic of persisting models in different ways. Connected Data Objects (CDO, http://www.eclipse.org/cdo/) can be used to store EMF models in a central repository. Although the pluggable storage backend of CDO is very promising, the variety of supported storage types is very limited and covers mostly relational database systems. A similar approach regarding the persistence of models employs the framework Teneo. It provides a model-relational mapping using Hibernate and EclipseLink. Teneo is also used in the CDO Hibernate Store and supports only relational databases.

A very interesting EMF-based framework, especially considering the resource layer which is discussed in the next section, is Texo. From a model definition it builds a web server which uses the Java Persistence API (JPA) (Oracle Corporation, 2013c) to store model data in relational data stores. In addition it provides a REST interface which allows clients to retrieve and modify data objects over a network. For this purpose the data objects are serialized in XML or JSON. Unfortunately it also does not fulfill the requirement that different types of data storage systems, e.g. document-oriented or cloud storage systems, should be supported. Therefore we decided to develop an own flexible solution with JDO and DataNucleus respectively.

4.5 Resource Layer

The resource layer should provide a programming language-independent interface in order that different applications can access data objects in a simple and uniform way. As solution we chose a REST-based Web service (Richardson and Ruby, 2007). We decided to use REST over a SOAP-based Web service (W3C, 2007) because the processing of domain data will follow the CRUD functionality which is represented by the standard HTTP methods (GET, PUT, POST, DELETE). Furthermore the SOAP protocol has a significant overhead due to increased data volume transferred by the service. REST services with Java are defined by the JAX-RS specification (Oracle Corporation, 2013d). We used Jersey (http://jersey.java.net/) for the implementation of the resource layer because it is the reference implementa-

tion of the JAX-RS specification which is widely used by well-known Java projects.

4.6 Representation Layer

To combine the JDO technology with the REST service, we needed to transform Java objects into a language-independent representation format, e.g. XML or JSON. A standard for this purpose is JAXB (Java Architecture for XML Binding) (Oracle Corporation, 2013a) which can be used for binding XML data to Java objects and vice versa. Thus, object instances of the representation layer have dependencies on the model and resource layer and transform the incoming data from the application layer to the corresponding JDO objects of the storage layer.

4.7 Technology Stack

An overview of the chosen technologies is given in the technology stack in Fig. 4. The stack has five layers of components that we used to build a toolkit for generation of Data Access Services. The layers of the stack have a fixed sequence owing to the dependencies between the layers. The sequence of the layers gives rise to the specific build process that is explained in Section 5.2. The first layer includes the definition of new domain-specific models done with a simple graphical editor based on EMF or a more complex diagram editor created with GMF. Acceleo templates are used to transform the model definition to concrete Java source code for the kernel API which is built with implementations of the specifications for JDO, JAXB and JAX-RS. Then an Eclipse plug-in compiles the kernel API to concrete Java bytecode. Finally the compiled kernel code is packaged and deployed in a lightweight runtime environment for Web services, for example Apache Tomcat or Jetty from the Eclipse Foundation, using Ant.

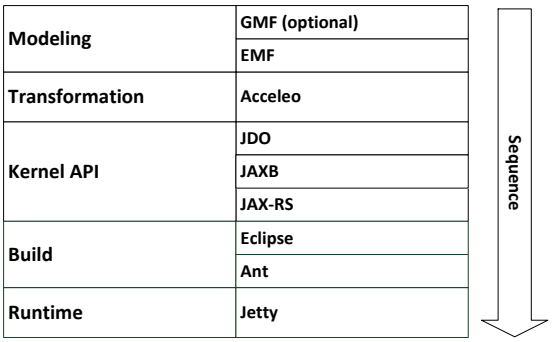

Figure 4: The technology stack used to build the toolkit.

5 TOOLKIT IMPLEMENTATION

One of the objectives is to provide a modeling environment which supports domain experts with graphical editors for creating their models including a self-explanatory GUI to manage the build process of the Service Generator and to configure the application and storage access layers. To tackle this task we have implemented a toolkit based on the Eclipse platform which in turn is based on the Equinox OSGi (http://www.eclipse.org/equinox/) implementation and available for many operating systems. All toolkit components were developed as Eclipse plug-ins and made available on an Eclipse update site so that users can install the toolkit with a few clicks into their Eclipse installation.

The plug-ins can be classified in three categories: The first category are the plug-ins that are based on EMF and which are generated mostly automatically. The second category contains the generation plug-in which manages the model-to-text transformation process. Here the Acceleo templates are specified which are used to generate all source and configuration files of the Data Access Service from a domain model. The third category includes the main plug-in of the toolkit, which we will discuss as well as the EMF-based plug-ins in more detail below.

5.1 EMF-based Plug-ins

The first plug-in in this category specifies the Ecore representation of the metamodel. Additionally a generator model automatically has been derived by EMF from the metamodel which stores parameters for the EMF code generator used to create source code for other plug-ins described below. These two models form the base for additional GMF models (gmfgraph, gmftool, gmfmap, gmfgen) that are used to create a high-quality diagram editor.

A plug-in whose content is based on the generator model is the *edit* plug-in. Its classes define the base for graphical model editors. Some included icons are used to mark the different model elements in the editor view. To reduce the number of selectable choices for the data type of an *Attribute* we made some changes to this plug-in so that only the most common used types, e.g. Boolean, Integer, Float and String, are displayed in the editors.

Also based on the generator model an *editor* plug-in has been generated. It contains classes for an entire graphical editor consisting of multiple views (model view and properties view). Additionally the plug-in adds a wizard to the Eclipse platform which assists users with the creation of new model files. We did not

make any changes in the source code of this plug-in.

To provide a graphical model editor which is more comprehensive than the one defined before we built a *diagram editor* plug-in with GMF. The resulting diagram editor looks similar to a simple UML class editor and facilitates a more sophisticated overview of the data entities and the connections between them.

5.2 Main Plug-in

The main plug-in references the others to provide an intuitive graphical interface and hides the details of the generation process of the Data Access Service from the users.

With the help of wizards users can create a new project in Eclipse which contains a default model file as well as a basic properties file which are starting points for the users to define a new domain model. When a model is designed the user can execute a wizard which will guide them through the build process. The user can choose if only a web application file should be generated or a full-blown ready-to-start web server with the web application already installed. To make this possible we distribute a Jetty web server with the toolkit. Although we provide Jetty by default, the generated web application is runnable on different servlet containers or Java application servers, e.g. Apache Tomcat or GlassFish.

A wizard parametrizes and manages the build process for the Data Access Service. The user has to enter several properties regarding the storage and the Web service layer that are mandatory to create a functional kernel. If the user enters valid parameters the wizard executes a sequential workflow which builds the Data Access Service with the help of Ant.

Figure 5 shows the build process in more detail. The initialization step creates a folder structure for the generic Java source code, properties files and compiled classes. Then the code generation with Acceleo from the user-generated domain model and a properties file is triggered. The generated Java classes are stored in a package structure which reflects the kernel architecture presented in Fig. 3. The usage of DataNucleus implies an enhancement step which is necessary to extend the byte code of the classes from the resource package with additional functionality needed for persistence. In the deployment phase the compiled and enhanced classes, constituting the ready-to-run Data Access Service, are packed into a WAR archive. Finally the WAR file and additional storage drivers are copied to a Jetty web server which is then zipped.

6 BENEFITS OF THE DATA ACCESS SERVICE

In this section we discuss some aspects in our model-driven Data Access Service that have been addressed to obtain acceptance in the scientific community and the industry.

6.1 Deployment

The Service Generator produces a standard WAR file (Web application ARchive) containing the generated kernel classes and configuration files that constitute a web application which can be deployed on different application servers. We have deployed and run the generated WAR file successfully on an Apache Tomcat and a Jetty web servers. These servers can host multiple instances of Data Access Services in parallel. To increase the automation, the Service Generator has the ability to create a ready-to-use Jetty web server containing the web application together with a bunch of tested storage drivers.

6.2 Security

Security is always important when processing user data especially when the service is available via the Internet. Application servers provide a lot of functionality in the area of security which covers encryption, authentication and authorization. All web servers support data encryption via the SSL protocol. Since we are using REST technology all aspects of authentication and authorization can be handled via Basic-Authentication or Digest-Authentication which are also supported well by the Web server vendors. Beside that alternative authentication protocols, e.g. Shibboleth, Public Key or Kerberos, can be used to protect the service from unauthorized parties.

6.3 Evolutionary Design

Usually domain models are subject to permanent changes. This increases the effort for developing scientific workflows because changes in the domain model require changes on all components of the simulation infrastructure in particular on the client and the storage side. Changes on the client side can be minimized by generating the client code using the Web service description that is automatically published by the Data Access Service. Our major advantage of using JDO in the persistence layer is the support of schemaless storages, e.g. the NoSQL database MongoDB. Such databases keep untouched when changing the domain model because the structure of the

Figure 5: Build process for the Data Access Service.

stored documents can vary. However DataNucleus can handle changes on the domain model even for relational databases as long as the model changes are limited to the addition of new entities or attributes. DataNucleus will add new tables and columns automatically to the database schema. However, after deletion or change of an existing entity or attribute the database schema must be updated manually.

6.4 Service Access

The implementation of the application access layer as a RESTful service allows access from applications written in various programming languages which have HTTP support. Furthermore the transport format is based on XML or JSON which are also in common use for many programming languages. The REST service is described in a WADL file (Web Application Description Language) including the URIs of the service resources. An XSD schema file, also provided by the service, describes the XML representation that is used to transfer the data entities between a client application and the service. In the Java environment, the Jersey Client API and the JAXB API can be used to create service stubs for clients basing on the WADL and XSD files. On the one hand this reduces the effort of programming client code to a minimum and on the other hand it even makes the integration of some applications containing many components feasible at all.

Applied to constructing workflows from standard components the Data Access Service provides a better alternative to the common practice of using format converters between workflow steps by replacing m:n transformations by a 1:n transformation. Even existing tools, which operate for a limited set of supported formats, will benefit in such a way that they do not have to exchange data in any other formats. Thereby, the only remaining data transformation is the one between the internal representation and the one imposed by the data model.

6.5 Performance

An important factor for the acceptance of the toolkit is the performance of reading and writing data entities via the service. To get some insight we have carried out performance measurements for three different storage types, a document-based MongoDB, a relational database system PostgreSQL and cloud storage Amazon S3 as persistence backends. For this purpose we defined an example entity type named *Atom* representing an atom with some attributes such as position coordinates, charge and element type. The performance test was done with a Jersey client that wrote Atom entities in XML format via HTTP POST requests to the storage. Beside the number of transferred entities we distinguished whether the entities are transferred bundled in one POST request (list) or sequentially (seq) in several POST requests. With these measurements we could examine the overall effort for writing data from the application to the storage as displayed in Fig. 6.

For all measurements MongoDB outperforms the PostgreSQL. For up to 10000 atom entities all write methods scale almost linearly. Then there is a crossover point after which a steeper linear region is observed. The difference between the MongoDB and PostgreSQL is smaller as between the single and the bundled write modes, i.e. the effect of the choice between MongoDB and PostgreSQL is minor. Furthermore, MongoDB seems to scale better with increasing number of entities for single writes (the corresponding curves diverge in Fig. 6). In contrast, for bundled writes, MongoDB and PostgreSQL seem to converge and for sufficiently large data might have the same performance. Using the Amazon S3 storage, which is about 100 times slower than the slowest local storage and write mode (PostgreSQL/seq), the bundled transfer of data entities is always more efficient. The low performance is due to the used implementation of JDO (DataNucleus) which executes additional service requests to Amazon over the network. The bottleneck seems to be either the network connection, the structure of the network communication or constraints of the Amazon service. The possibility to choose between different storages depending on usage scenario is a clear benefit of the Data Access Service.

7 RELATED WORK

In this section we outline some recent developments which have much in common with our present work. We are not aware of an environment satisfying all requirements of the user communities as discussed in

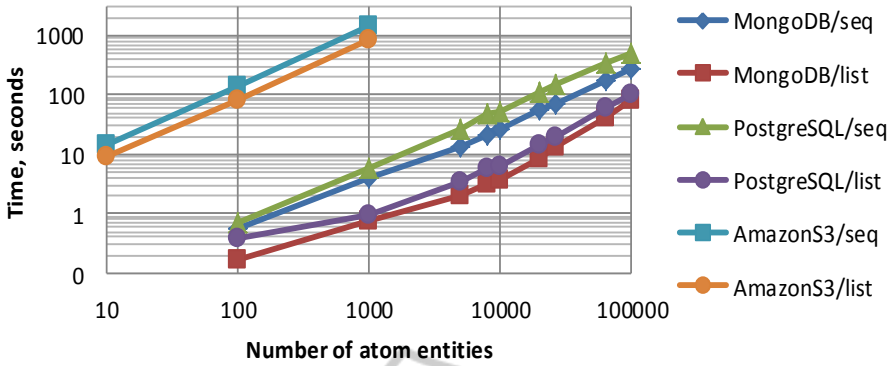

Figure 6: Throughput performance of the Data Access Service measured for three different storage backends.

Section 2. In specific scientific and engineering domains (for instance, in materials science) the practical benefits of the model-driven engineering and service-oriented architecture is still limited. Existing solutions, some of which are described in the following, are either too generic, and hence inaccessible for such communities, or too specific and difficult to transfer to other application scenarios.

The MEMOPS (MEta-MOdel Programming System) (Fogh et al., 2010) code generation machinery is a good example for a graphical modeling tool created by the Collaborative Computational Project for NMR (CCPN) (Vranken et al., 2005) and deployed in the domain of the nuclear magnetic resonance. MEMOPS offers a graphical tool where domain specialists can design their models in the UML (Unified Modeling Language) notation. The MEMOPS framework creates data access libraries (via APIs) and data storage implementation automatically from the model description. Unfortunately, the generated APIs are restricted to Python, Java and C that limits the number of applications and the only supported storage types are local XML files and SQL databases. Therefore, the system is limited to non-distributed applications, i.e. running locally. In another approach for biomolecular NMR data analysis, workflow models and conceptual and logical data models (Ellis et al., 2006) have been proposed which have led recently to the CONNJUR integrated environment (Nowling et al., 2011) aiming to support the entire process of molecular structure determination.

In the framework of the Integrated Tokamak Modeling Task Force (ITM), the Universal Access Layer (UAL) (Manduchi et al., 2008) has been developed to provide capability of storing and retrieving data involved in a simulation using the Kepler workflow system. The underlying hierarchical data structure is based on the storage formats MDSplus and HDF5 and the granularity in data access is defined by a set of so-called Consistent Physical Objects.

The SIMPL framework architecture has been proposed for access to heterogeneous data sources in scientific workflows (Reimann et al., 2011). The SIMPL framework has been designed as extension to existing scientific workflow management systems to provide abstraction for data management and data provisioning in scientific and engineering simulation workflows. However, the SIMPL framework does not provide means for meta-modeling and data models for the data structures.

Recently an approach called Morsa (Espinazo Pagán et al., 2011) has been proposed for scalable access to large models through load on demand in which model persistence has been realized using a NoSQL database. Performance benchmarks with a prototype for EMF have shown performance superior to CDO and XMI especially in respect with reduced memory footprint achieved by partial loading of the data. More recently, the proposed language $T_{\square}$ (Rabbi and MacCaull, 2012) enables model-driven development and generation of multi-component workflow applications. Thereby, the aspects of the persistence component have been less emphasized as in our present work. Rather, the provided elaborated language syntax allows for implementing procedural statements, ontology queries, declaring user interfaces, applying access control policy, and task scheduling via Web service based access interfaces for client applications and resources.

MDE Eclipse tools have been employed for industrial development of distributed scientific workflow applications in the oil and gas domain (Rahon et al., 2012). Similar to the approach in our present work, the EMF/Ecore is used for modeling and Acceleo for code generation. Nevertheless, the realization does not seem to allow the same variety of storage backends and does not provide a language-independent Web service as client application interface but a C++ library API.

8 CONCLUSIONS

In this paper we presented a generic framework for model-driven management of data in composite scientific and engineering applications. The essential result of the realization of the framework is an EMF-based toolkit, consisting of a set of Eclipse plugins, which enables domain experts to develop data models and automatically generate self-contained data access services. For this purpose we defined as a specialization of the Ecore meta-metamodel a custom metamodel which is optimized for handling domain-specific pure data models. A data access service is automatically created employing a generative approach based on Acceleo and integrating JDO and JAX-RS to the service kernel, containing a persistence layer, a resource layer and a representation layer. The data is then accessible from the individual client application components via a language-independent Web service interface of the data access service.

The proposed solution is especially suitable for applications with high heterogeneity and complexity of data representations, diversity of programming languages of the integrated components and data storage resources used. The solution makes possible an evolutionary design of dynamically changing applications. Thus the toolkit can be used in all application domains of computational science for rapid development of complex dynamic applications and effective deployment as Web services. Although the toolkit strongly reduces the technical burden of data modeling and management, it can be combined with ontology-based frameworks such as the Apache Jena framework (http://jena.apache.org/) to tackle model complexity even more effectively.

Future work will focus on analysis of the scalability of the data access service, in particular considering applications for data-intensive analyses and evaluation in production environments. Additionally we envisage exploitation of the modeling approach to steer throughput performance optimizations. Also further practical aspects such as model revisions and model changes during service operation will be investigated.

As soon as the framework is released we will provide tutorials and example use cases to demonstrate the operability.

ACKNOWLEDGEMENTS

This work has been partially funded by the 7th Framework Programme of the European Commission within the Research Infrastructures with grant agreement number RI-261594, project MMM@HPC.

REFERENCES

Ambler, S. W. (2012). Mapping Objects to Relational Databases: O/R Mapping In Detail. http://www.agiledata.org/essays/mappingObjects.htm [Online; accessed 6-August-2013].

Bender, A., Poschlad, A., Bozic, S., and Kondov, I. (2013). A Service-oriented Framework for Integration of Domain-specific Data Models in Scientific Workflows. *Procedia Computer Science*, 18:1087 – 1096. 2013 International Conference on Computational Science.

Birkenheuer, G., Blunk, D., Breuers, S., Brinkmann, A., dos Santos Vieira, I., Fels, G., Gesing, S., Grunzke, R., Herres-Pawlis, S., Kohlbacher, O., Kruger, J., Lang, U., Packschies, L., Muller-Pfefferkorn, R., Schafer, P., Steinke, T., Warzecha, K.-D., and Wewior, M. (2012). MoSGrid: efficient data management and a standardized data exchange format for molecular simulations in a grid environment. *Journal of Cheminformatics*, 4(Suppl 1):P21.

Bozic, S. and Kondov, I. (2012). Dataflow Management: A Grand Challenge in Multiscale Materials Modelling. In Cunningham, P. and Cunningham, M., editors, *eChallenges e-2012 Conference Proceedings*, page Ref. 38. IIMC International Information Management Corporation.

Bozic, S., Kondov, I., Meded, V., and Wenzel, W. (2012). UNICORE-Based Workflows for the Simulation of Organic Light-Emitting Diodes. In Huber, V., Müller-Pfefferkorn, R., and Romberg, M. R., editors, *UNICORE Summit 2012 Proceedings, May 30-31, 2012, Dresden, Germany*, volume 15 of *IAS Series*, pages 15–25. Forschungszentrum Jülich GmbH Zentralbibliothek, Verlag.

Dubitzky, W., McCourt, D., Galushka, M., Romberg, M., and Schuller, B. (2004). Grid-enabled data warehousing for molecular engineering. *Parallel Computing*, 30(9-10):1019–1035.

Ellis, H., Fox-Erlich, S., Martyn, T., and Gryk, M. (2006). Development of an Integrated Framework for Protein Structure Determinations: A Logical Data Model for NMR Data Analysis. In *Third International Conference on Information Technology: New Generations, 2006. ITNG 2006.*, pages 613–618.

Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Espinazo Pagán, J., Sánchez Cuadrado, J., and García Molina, J. (2011). Morsa: A Scalable Approach for Persisting and Accessing Large Models. In Whittle, J., Clark, T., and Kühne, T., editors, *Model Driven Engineering Languages and Systems*, volume 6981 of *Lecture Notes in Computer Science*, pages 77–92. Springer Berlin Heidelberg.

Fogh, R. H., Boucher, W., Ionides, J. M. C., Vranken, W. F., Stevens, T. J., and Laue, E. D. (2010). MEMOPS: Data modelling and automatic code generation. *J. Integr. Bioinform.*, 7(3):123–145.

Fogh, R. H., Boucher, W., Vranken, W. F., Pajon, A., Stevens, T. J., Bhat, T. N., Westbrook, J., Ionides, J.

M. C., and Laue, E. D. (2005). A framework for scientific data modeling and automated software development. *Bioinformatics*, 21(8):1678–1684.

Kondov, I., Maul, R., Bozic, S., Meded, V., and Wenzel, W. (2011). UNICORE-Based Integrated Application Services for Multiscale Materials Modelling. In Romberg, M., Bala, P., Müller-Pfefferkorn, R., and Mallmann, D., editors, *UNICORE Summit 2011 Proceedings, 7-8 July 2011, Torun, Poland*, volume 9 of *IAS Series*, pages 1–10, Jülich. Forschungszentrum Jülich GmbH Zentralbibliothek.

Manduchi, G., Iannone, F., Imbeaux, F., Huysmans, G., Lister, J., Guillerminet, B., Strand, P., Eriksson, L.-G., and Romanelli, M. (2008). A universal access layer for the Integrated Tokamak Modelling Task Force. *Fusion Engineering and Design*, 83(2-3):462 – 466.

Murray-Rust, P., Townsend, J., Adams, S., Phadungsukanan, W., and Thomas, J. (2011). The semantics of Chemical Markup Language (CML): dictionaries and conventions. *Journal of Cheminformatics*, 3(1):43.

Nowling, R., Vyas, J., Weatherby, G., Fenwick, M., Ellis, H., and Gryk, M. (2011). CONNJUR spectrum translator: an open source application for reformatting NMR spectral data. *Journal of Biomolecular NMR*, 50:83–89.

OMG (2003). UML 2.0 Infrastructure Specification. Technical Report ptc/03-09-15, Object Management Group.

Oracle Corporation (2013a). JSR 222: Java(TM) Architecture for XML Binding (JAXB) 2.0. http://jcp.org/en/jsr/detail?id=222 [Online; accessed 6-August-2013].

Oracle Corporation (2013b). JSR 243: Java(TM) Data Objects 2.0 - An Extension to the JDO specification. http://jcp.org/en/jsr/detail?id=243 [Online; accessed 6-August-2013].

Oracle Corporation (2013c). JSR 317: Java(TM) Persistence 2.0. http://jcp.org/en/jsr/detail?id=317 [Online; accessed 6-August-2013].

Oracle Corporation (2013d). JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services. http://jcp.org/en/jsr/detail?id=339 [Online; accessed 6-August-2013].

Rabbi, F. and MacCaull, W. (2012). $T_{\square}$: A Domain Specific Language for Rapid Workflow Development. In France, R. B., Kazmeier, J., Breu, R., and Atkinson, C., editors, *Model Driven Engineering Languages and Systems*, volume 7590 of *Lecture Notes in Computer Science*, pages 36–52. Springer Berlin Heidelberg.

Rahon, D., Gayno, R., Gratien, J.-M., Le Fur, G., and Schneider, S. (2012). Migration to model driven engineering in the development process of distributed scientific application software. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, SPLASH '12, pages 181–190, New York, NY, USA. ACM.

Reimann, P., Reiter, M., Schwarz, H., Karastoyanova, D., and Leymann, F. (2011). SIMPL - A Framework for Accessing External Data in Simulation Workflows. In für Informatik (GI), G., editor, *Datenbanksysteme für Business, Technologie und Web (BTW 2011), 14. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), Proceedings, 02.-04. März 2011, Kaiserslautern, Germany*, volume 180 of *Lecture Notes in Informatics (LNI)*, pages 534–553.

Richardson, L. and Ruby, S. (2007). *RESTful Web Services*. O'Reilly, first edition.

Sadiq, S., Orlowska, M., Sadiq, W., and Foulger, C. (2004). Data flow and validation in workflow modelling. In *Proceedings of the 15th Australasian database conference*, volume 27 of *ADC '04*, pages 207–214, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Schmidt, D. (2006). Guest Editor's Introduction: Model-Driven Engineering. *IEEE Computer*, 39(2):25–31.

Schneider, O., Fogh, R. H., Sternberg, U., Klenin, K., and Kondov, I. (2012). Structure Simulation with Calculated NMR Parameters — Integrating COSMOS into the CCPN Framework. In Gesing, S., Glatard, T., Krüger, J., Olabarriaga, S. D., Solomonides, T., Silverstein, J. C., Montagnat, J., Gaignard, A., and Krefting, D., editors, *HealthGrid Applications and Technologies Meet Science Gateways for Life Sciences*, volume 175 of *Studies in Health Technology and Informatics*, pages 162–172. IOS Press.

Sild, S., Maran, U., Lomaka, A., and Karelson, M. (2006). Open Computing Grid for Molecular Science and Engineering. *J. Chem. Inf. Modeling*, 46:953–959.

Sild, S., Maran, U., Romberg, M., Schuller, B., and Benfenati, E. (2005). OpenMolGRID: Using Automated Workflows in GRID Computing Environment. In Sloot, P., Hoekstra, A., Priol, T., Reinefeld, A., and Bubak, M., editors, *Advances in Grid Computing - EGC 2005*, volume 3470 of *Lecture Notes in Computer Science*, pages 464–473. Springer.

Vranken, W. F., Boucher, W., Stevens, T. J., Fogh, R. H., Pajon, A., Llinas, M., Ulrich, E. L., Markley, J. L., Ionides, J., and Laue, E. D. (2005). The CCPN data model for NMR spectroscopy: Development of a software pipeline. *Proteins: Structure, Function, and Bioinformatics*, 59(4):687–696.

W3C (2007). SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).