Model-Driven Migration of Scientific Legacy Systems to Service-Oriented Architectures

Jon Oldevik, Gøran K. Olsen
SINTEF Information and Communication Technology
Forskningsvn 1, 0373 Oslo, Norway
jon.oldevik | goran.olsen at sintef.no

Ute Brönner, Nils Rune Bodsberg
SINTEF Materials and Chemistry
Brattørkaia 17c, 7465 Trondheim, Norway
ute.broenner | nilsrune.bodsberg at sintef.no

***Abstract*—We propose a model-driven and generative approach to specify and generate web services for migrating scientific legacy systems to service-oriented platforms. From a model specification of the system migration, we use code generation to generate web services and automate the legacy integration. We use a case study from an existing oil spill analysis application developed in Fortran and C++ to show the feasibility of the approach.**

***Keywords*-Model-driven engineering, legacy migration, web services**

I. Introduction

A large number of existing systems, especially within data and computationally intensive domains, are based on implementations that are becoming increasingly difficult to maintain and evolve [1], typically in languages like Cobol and Fortran. Competent personnel with knowledge of these technologies is also becoming a scarce resource. Modernisation toward a service-oriented architecture may also open for new business opportunities.

In this paper, we investigate a model-driven approach for migrating legacy systems to service-oriented architectures. Our migration strategy is wrapping of existing legacy components. We use the Unified Modelling Language (UML) to specify migration models, or wrappers, that are fed to model-driven code generators to generate a deployable service. This work has been done in the SiSaS project[1], which has an overall focus of methods and tools for migrating *scientific software* to service-oriented architectures.

We define a migration profile in UML that contains concepts for integrating with existing legacy, such as native libraries, executable programs, and databases, as well as for integrating with existing web services. We establish a modelling approach – a method – for how to specify services using the migration concepts, as well as concepts from SoaML [2]. Our modelling comprises the interfaces and structure of a service, as well as the behaviour of different service parts.

Our goal is to create effective and usable means for migrating legacy systems to service-oriented platforms. Our conjecture is that model-driven and generative techniques can provide these means.

[1]SINTEF Software as a Service

II. Motivating Case Study: Oildrift Simulation

In SINTEF Materials and Chemistry, they have a commercial legacy product for simulating oil drift, which can help predicting the spreading of oil in case of an accidental spill. The system is implemented by a Fortran simulation back-end and a C++ front-end. Now, they want a transition to a service-oriented paradigm to more easily adapt to new customer needs and more flexible business models. Figure 1 illustrates the existing application.

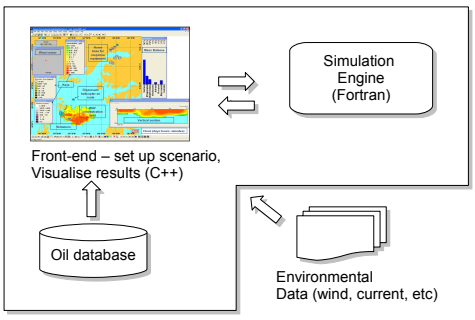

Figure 1. Oildrift Prediction – Legacy Application

The Fortran simulation core is responsible for simulating oil drift based on numerical models. It is invoked from a presentation layer written in C++. All input is file based, and simulation runs in batch mode from some minutes to several days. This approach has worked fine for many years, but there are some apparent challenges with respect to interoperability, integration, and scalability. The goal is to migrate the application to meet new market needs while coping with these challenges.

III. Our approach

We use model-driven engineering techniques to develop the oil drift prediction as a service that wraps the existing simulation engine. UML models are used to specify the service interfaces and the details of the wrapper architecture. From these models, we generate

XML schemas for the web service, Java interface and class implementations of the web service, the architecture of the wrapper, and its behavioural implementation. Wrapping of the C++ front-end is out of our scope, since this will be re-designed to fit a web-based interaction paradigm.

We define a UML *migration profile* to represent semantics of different types of migration features, such as *executables*, *databases*, and *native libraries*. The code generators use this semantics to generate the necessary integration code. Figure 2 illustrates the high-level approach.

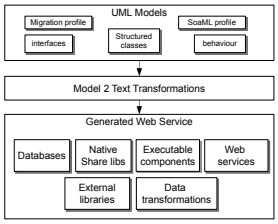

Figure 2. Approach Overview

We use UML interfaces and classes to model the structural parts of the system. Service interfaces define the behaviour, and classes define the internal structure of the services. UML composite structures are used for specifying the service de-composition into parts. The service is decomposed by legacy component parts, which is orchestrated by the service to provide its operations. The behaviour of the service and its contained legacy wrapper components is defined by UML activity diagrams.

To relate the migration models to the service-oriented modelling domain, we use some SoaML concepts to describe services: the stereotype «serviceInterface» is used to denote a service, i.e. the service that wraps the legacy systems. The stereotype «MessageType» is used to specify the data types passed as message input and output of the service.

A. The migration profile

The migration profile contains a set of stereotypes used for adding migration semantics to the UML models. The main purpose of the migration model is to integrate existing legacy functionalities and expose them through well-defined interfaces. To this end, we use standard UML models extended with migration semantics from a profile.

The Component Types: The component types represent different sorts of legacy components that take part in fulfilling the responsibilities of the legacy system. This might be existing shared libraries, executables, Java libraries, databases, or web services. Figure 3 specifies the set of component types that are in the current profile.

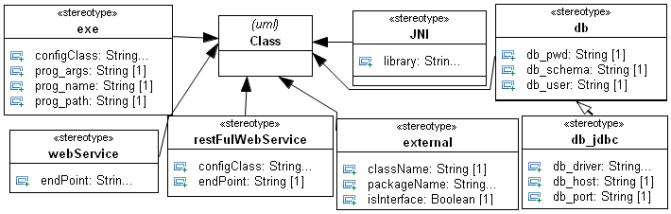

Figure 3. Component types

The stereotype «WebService» denotes the wrapping of an external web service, i.e. a web service client. «RestfulWebService» denotes the wrapping of a restful web service. Its endpoint is an URI that acts as a data source, which is fetched by the wrapper and used locally. «exe» denotes the wrapping of an executable program. «JNI» denotes the wrapping of a native library, such as a windows shared DLL, using Java Native Interface (JNI). «external» denotes integration with external Java libraries, e.g. provided by a *jar* file. «db_jdbc» denotes the wrapping of a JDBC database. Operations defined in classes of this type represent database SQL queries.

The profile additionally provides stereotypes for specific types of operations, such as «asynch» for *asynchronous* operations and «RSOp» for restful service operations. Exceptions can be specified explicitly by classes stereotyped «exception». Throwing and catching of exceptions are specified by dependencies stereotyped «throws» and «catches».

Behaviour – Activities and Actions: Behaviour is declared by operations in components. The behaviour of these operations are defined by associated *activity diagrams*. An activity diagram defines behaviour by sequences and branches of actions that are mapped to statements in code generation. Standard (opaque) actions contain embedded Java code. *CallOperationActions* are used for defining invocations to defined operations of related components. In addition, we define a set of stereotypes in the migration profile for simplifying the action specification:

«return» is used to denote a return from the method execution with a specific value; «assign» is used to denote an assignment of a value to a variable; «setState» is used to denote the setting of an internal state variable, specifically used for asynchronous and long running operations; «setReturn» is used to set the return value of an asynchronous and long running operation; «param» is used to define an input parameter to a *CallOperationAction*. It references a previously defined variable;

finally, «valueparam» is used to define a literal value as an input parameter to a *CallOperationAction*.

B. Modelling the Oildrift Prediction Case

In this section, we exemplify the use of the migration profile on the oildrift prediction case in terms of structure and behaviour.

Service Structure and Interface: The service itself is defined by a SoaML «serviceInterface» class, which implements the service interface with a set of exposed operations (Figure 4). The most interesting of these is the «asynch» operation *predictOilDriftAsynch*, which provides the main service in the oildrift prediction case. Since the execution of a simulation may run for hours, or even days, the operation is declared as asynchronous. The operation will return immediately with only a *session id* to identify the session.

Two additional helper operations are provided for checking execution status (*getStatus*) and to retrieve the result upon termination (*getPredictOilDriftResult*).

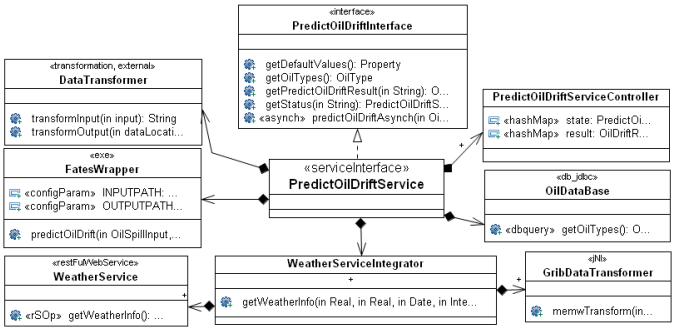

Figure 4. Predict Oildrift – Service and Wrapper Components

The *PredictOilDriftService* is a structured class that contains a set of parts: the *PredictOilDriftServiceController* is the internal orchestration component for the service. All incoming calls are delegated to the controller, which implements the operations of the service. The *DataTransformer* provides operations for transforming input required by the *Fortran* simulation engine, and transforming result data after simulation. The *FatesWrapper* is an «exe» component, which wraps the execution of the *Fortran simulation* program. The *WeatherServiceIntegrator* provides operations for integrating with an external weather data provider. It is further de-composed by two parts: a «restfulWebService» called *WeatherService* and a «JNI»-component called *GribDataTransformer*. The *getWeatherInfo* operation gets weather data from a restful web service that provides binary data in the *GRIB*[2] format. To transform the GRIB-data to the input format used by the simulation engine, an external native library (in this case DLL) is integrated by the *GribDataTransformer*. The *OilDatabase* is a «db_jdbc»-component, which provides oil type information from an SQL database.

The *data types* passed in the service interface are modelled as classes stereotyped using the SoaML stereotype «MessageType». Apart from the stereotype, the data types are specified with standard UML classes with attributes and associations.

Behaviour: Component behaviour is specified for different operations in the migration model. Behaviour is defined by *Activity diagrams* that are associated with the operations they implement.

Figure 5 shows the behaviour of the *predictOilDriftAsynch* operation, which is contained in the *PredictOilDriftServiceController*. All invocations to the service *PredictOilDriftService* is by convention delegated to its *controller* part.

The example with the asynchronous operation is interesting, since it also requires handling of the long-running external executable. In this case, this is handled by providing a second activity diagram, which specifies the behaviour upon termination of the executable.

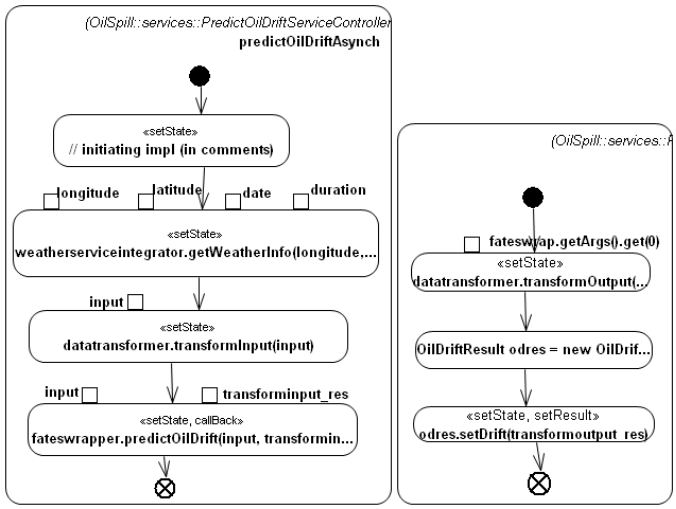

Figure 5. Predict Oildrift Service Behaviour

The first activity diagram specifies the initial behaviour of the operation, up until the call to the *executable* component.

The second activity diagram from Figure 5 specifies the behaviour occurring when the execution of the external program has terminated. When the final action is executed, the service state (for this particular session id) is set to a *ready* state, and the client can fetch the *result* of the service.

Our approach for behavioural modelling is tied to the target language, in this case Java, since we in some cases embed small portions of code inside actions. We

[2]GRIdded Binary, http://www.wmo.int

could get around this by incorporating a richer set of UML actions that can cope with variable declarations, property references, and object creation. This extension will be investigated as part of future work.

C. Code generation

The purpose of the migration models is to automate as much as possible the legacy system migration process. We have developed a set of transformations, or code generators, to support the transition from models to deployable services. They were implemented with the model to text transformation tool MOFScript[3].

IV. Related Work

Dorda et al.[4] give a survey of legacy system modernisation approaches. They distinguish between two main types of modernisation: *white-box* and *black-box* modernisation. White box modernisation requires an understanding of the internal parts of a system, and involves re-structuring, re-architecting, and re-implementing the system. Black-box modernisation is only concerned with the input/output, i.e. the interfaces, of the legacy system, and is often based on wrapping. Our approach can be seen as a model-driven black-box modernisation technique. However, the migration also has flavours of white-box migration to it, in particular in understanding and transforming the legacy data formats.

Within the Object Management Group (OMG), the Architecture-Driven Modernization (ADM) task force [5] is working on standards to support legacy modernisation, such as meta-models for knowledge discovery, software visualisation, and refactoring.

Razavian and Lago [6] present a SOA migration framework – (SOA-MF) – wherein they establish an overall process framework for legacy migration, focusing on recovery and re-engineering, and put it in the context of migration methods such as SMART [7]. Although our work has not focused on re-factoring or re-engineering, the processes targeting legacy discovery and transformation to new architecture has also been addressed in our work.

Canfora et al.[8] present an approach for migrating interactive legacy systems to web services based on wrapping. They define a model (a state machine) of the user interactions, which is the basis for integration with legacy systems through *terminal emulation*. This process is then exposed as a web service.

V. Conclusion and Outlook

We have presented a model-driven approach for legacy migration to service-oriented architectures, where the focus is black-box migration by wrapping legacy components using model-driven and generative techniques. We have defined a UML profile for migration and a set of code generators for generating services and wrapper components. We have tested the migration approach on an oildrift prediction system, by modelling and generating the services and wrappers required for integration with the various legacy components.

At this time, we have not addressed *automation of legacy data mappings*, which is a major concern in legacy modernisation. In the current case study, data mappings between binary data formats where written manually. In future work we will investigate appropriate techniques and tools for specifying data transformations at the model level, and for mapping these to the implementation level.

Acknowledgements

The results reported in this paper are from the SiSaS project *(SINTEF Software as a Service)*. SiSaS is an internal project within SINTEF that focuses on migration of scientific legacy scientific software to services.

References

[1] F. Zoufaly, "Issues and challenges facing legacy systems," *Project Management, developer.com*, 2002.

[2] Object Management Group (OMG), "Service Oriented Architecture Modeling Language (SoaML), FTF Beta 2," OMG, Standard ptc/2009-12-09, 2009.

[3] J. Oldevik, T. Neple, R. Grønmo, J. Aagedal, and A. Berre, "Toward Standardised Model to Text Transformations," in *European Conference on Model Driven Architecture - Foundations and Applications (ECMDA)*. Nuremberg: Springer, 2005, pp. 239–253.

[4] S. Comella-Dorda, K. Wallnau, R. C. Seacord, and J. Robert, "A survey of legacy system modernization approaches," 2000. [Online]. Available: http://handle.dtic.mil/100.2/ADA377453

[5] Object Management Group (OMG), "ADM White Paper: Transforming the Enterprise," OMG, White paper http://www.omg.org/docs/admtf/07-12-01.pdf, 2008.

[6] M. Razavian and P. Lago, "Understanding SOA migration using a conceptual framework," *Czech Society of Systems Integration*, 2010.

[7] S. Balasubramaniam, G. A. Lewis, E. J. Morris, S. Simanta, and D. B. Smith, "SMART: Application of a method for migration of legacy systems to SOA environments," in *Service-Oriented Computing - ICSOC 2008, 6th International Conference, Sydney, Australia, December 1-5, 2008. Proceedings*, ser. Lecture Notes in Computer Science, A. Bouguettaya, I. Krüger, and T. Margaria, Eds., vol. 5364, 2008, pp. 678–690.

[8] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana, "Migrating interactive legacy systems to web services," in *CSMR*. IEEE Computer Society, 2006, pp. 24–36.