UML2 Profile and Model-Driven Approach for Supporting System Integration and Adaptation of Web Data Mashups

Patrick Gaubatz and Uwe Zdun

Faculty of Computer Science
University of Vienna, Vienna, Austria
{firstname.lastname}@univie.ac.at

Abstract. From a system integration perspective, Web data mashups used in larger architectures often need to be integrated with other system components, such as services, business processes, and so on. Often a change in one of these components requires changes in many of the dependent components. Similarly, an analysis of some system properties requires knowledge about other system parts than just the mashup. Such features could be implemented using the model-driven development (MDD) approach, but existing MDD approaches for mashups concentrate on modeling and execution only. To remedy this problem, we propose a generic approach based on a UML2 profile which can easily be extended to model other system parts or integrated with other existing models. It is the foundation for generating or interpreting mashup code in existing languages as well as other system parts using the MDD approach and performing system adaptation or analysis tasks based on models in a standard modeling language.

1 Introduction

Web mashups are used to combine data from different Web documents and services to create new functionality. Web data mashups concentrate on extracting and transforming data from such Web data sources and offer them as a service. Different domain-specific languages (DSLs) that are tailored specifically to facilitate the development of Web mashups (see e.g. [1–4]), model-driven approaches for Web mashups and Web data integration [5, 6], and extensions of existing behavioral modeling languages like BPEL [7, 8] have been proposed to model Web mashups.

Most approaches today concentrate on mashup modeling and execution. From a system integration perspective, they offer two means for system integration: (1) they integrate data from Web documents and services and (2) they offer their results either as Web documents or services. The larger system integration or architectural context is usually not supported any further by current approaches. For instance, Web data mashups may be used to integrate data from various internal and external information systems. Changes (e.g. of the service interface) in any of these information systems might require adaptations of the dependent Web data mashups.

The model-driven development approach (see e.g. [9]) offers a convenient way to address this problem. Via a model-driven generator, we can generate different components

M. Grossniklaus and M. Wimmer (Eds.): ICWE 2012 Workshops, LNCS 7703, pp. 81–92, 2012.

from models and re-generate the code upon changes in the models. Via a model-driven interpreter we could even support model-based runtime (on-the-fly) adaptation of the mashups. Finally, the model-driven approach could be used to generate other representations of the models. For instance, we could generate a Petri Net or automata representations of the complete process and mashup behavior to analyses aspects like deadlocks or life-locks in the entire model.

From a modeling perspective, mashups are similar to areas like behavioral software modeling (see e.g. [10]) and business process modeling or workflows (see e.g. [11]). In essence, mashups can be seen as behavioral composition models similar to UML activity diagrams [12] or microflows [13] (a microflow is a short-running, non-persistent workflow [13]), with specific functionality such as extracting data from Web pages, invoking services, and combining the data retrieved from Web pages and services using scripts. Some modeling approaches that extend existing behavioral modeling languages like BPEL have been proposed [7, 8], but BPEL is designed for long-running, transactional business processes (macroflows) rather than short-running microflows.

In this paper, we propose a UML profile for mashup modeling that is based on a core package describing basic microflows as an extension of UML activity diagrams. Mashup-specific functions are added in an extension package. In this package we semi-formally modeled some of the most common mashup functionalities. The profile is designed so that it can be extended with more specific mashup functions that are provided by mashup approaches. The core contribution of this paper is a semi-formal profile for core mashup functionality as an extension of the UML2 meta-model. As a proof-of-concept we have also implemented a model-driven interpreter for the mashup profile. To explain the generalizability of our mashup modeling profile and show that it can serve as a unified modeling approach for many existing mashup approaches, we also discuss how our approach can be used in model-driven code generators to cover other existing mashup approaches.

2 Problem Description

Current Web data modeling approaches do not consider Web data mashups in a larger architectural context. For instance, the mashup may be used inside of a business process, and both mashup and process must be monitored. Figure 1 shows the architectural overview of this example scenario. In this simple architecture example, we must integrate the business process, the mashup, the used services, and the used Web sites, and provide monitoring rules for all these components as well as their deployment configurations. If we perform changes, all these artifacts might need to be changed. Keeping them consistent during development and maintenance is tedious and error-prone.

The model-driven development approach helps to overcome this problem. Unfortunately, using the model-driven approach with mashups is difficult as they are often described in proprietary modeling or script languages and there is no unified modeling approach for them that enables us to use model-driven development approaches together with mashup approaches. Standard modeling languages that provide convenient ways to model other system parts as well like the UML are usually not used (e.g. service interfaces can be modeled as extensions of class diagrams). Furthermore, the existing model-driven mashup approaches (e.g. [5, 6]) focus on specific aspects (like user

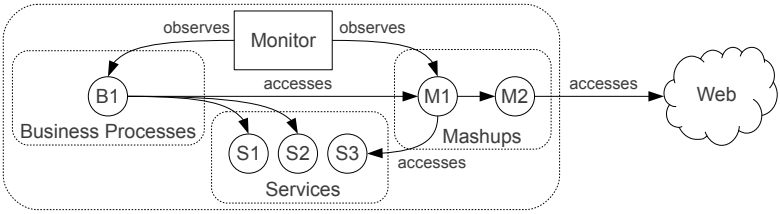

Fig. 1. Architectural Overview of a System Integration Scenario

interface layer integration) and offer only limited support for the integration of mashups in larger architectural contexts.

3 UML2 Profile for Modeling Web Data Mashups as Microflows

In order to model Web data mashups, different primitives, such as service invocations, transformation of data, and output generation in a mashup must be modeled and interconnected. To model such primitives we chose the profile extension mechanism of UML2 because there are already existing UML2 meta-classes that are semantically a close match to the characteristics of a Web data mashup. In particular, a mashup can be seen as a series of activities that perform data transformations. From the perspective of behavioral modeling, a mashup can be seen as a special purpose *microflow*: The term microflow refers to a short running, rather technical process model [13][1]. A typical way to model microflows are UML2 activity diagrams, which we will extend using a UML2 profile for modeling mashups as microflows.

This is done by semi-formally extending semantics of the respective UML2 meta-classes (rather than having to define completely new meta-classes). A profile is still valid, standard UML2. That is, it can be used in existing UML2 tools, instead of having to offer proprietary ones which are rarely used in practice. We use the Object Constraint Language (OCL) to define the necessary constraints for the defined stereotypes to precisely specify their semantics. OCL constraints are the primary mechanism for traversing UML2 models and specifying precise semantics on stereotypes.

Below, each primitive is precisely specified in the context of the UML2 meta-model using OCL constraints. This is a very important step for the practical applicability of our concepts: Without an unambiguous definition of the primitives, they cannot be used (interchangeably) in UML2 tools and model-driven generators. That is, our main reason for using the UML2 – a potential broad tool support – could otherwise not be supported.

3.1 Modeling Microflows

As a Web data mashup can be seen as a microflow, we decided to found our profile for Web data mashups on a meta-model extension for microflows. More precisely, we are proposing a meta-model for scripting language-based microflows in the context of service composition and service-based data integration.

[1] Microflows can be contrasted to macroflows which describe long-running, rather business-oriented process [13].

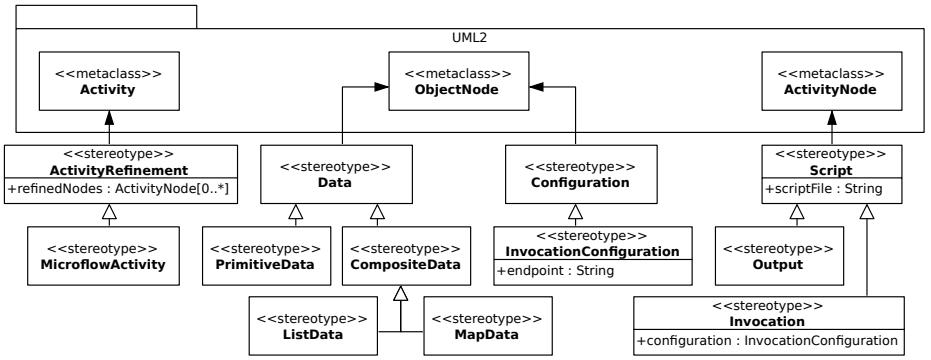

Fig. 2. The Microflow Meta-Model

Figure 2 depicts the UML2 class diagram of the microflow meta-model. The *MicroflowActivity* stereotype allows us to denote an UML2 activity to be a microflow. It also allows us to make the model subject to model constraints. For example, we defined an OCL constraint (see Listing 1) specifying that an instance of a microflow must have exactly one *InitialNode* – a requirement needed to allow the execution of microflows.

```
context MicroflowActivity
  inv: self.baseActitiy.node->select(oclIsTypeOf(InitialNode))->size() = 1
context Script inv: self.scriptFile->notEmpty()
context InvocationConfiguration inv: self.endpoint->notEmpty()
context Invocation inv: self.configuration->notEmpty()
context Output inv: self.baseActivityNode.incoming->exists(in |
  Data.allInstances()->exists(data |
    in.source.oclIsTypeOf(ObjectNode) and in.source = data.baseObjectNode))
```

Listing 1. OCL Constraints for the Microflow Model

Microflows of Web data mashups read, write, transform, process, analyze, annotate, group, ... data. Consequently, our meta-model defines a *Data* stereotype. In our approach, instances of *Data* are called data objects. *Data* can either be *PrimitiveData* (e.g. strings, numbers, or boolean values) or complex *CompositeData*. The latter can either be *ListData* (i.e. arrays) or *MapData* (i.e. key/value-pairs). These two complex data structures allow us to accommodate and map (at least) the two most widely used data formats in the Web context: XML and its variations (e.g. HTML) as well as JSON.

Having introduced data objects, we have yet to define means to get them into/out of a microflow. An *Output* returns data and/or a result (e.g. an XML document) back to the executor of the microflow (e.g. a Web application). An *Invocation* is used to retrieve data to be processed from a service (e.g. a RESTful Web service).

A *Script* acts as a "placeholder" for implementation-level code. This way arbitrary extensions from existing mashup implementation languages can be integrated – allowing us to model mashups in a generalizable fashion, but still being able to incorporate the specialized features of different mashup languages via code generation. That is, the model-driven interpreter or generator will take the code in the script files and insert it at the dedicated points into the generated or interpreted code. For this reason, *Script* serves both as the meta-model's primary extension point and as a "fallback" activity. Although the meta-model is extensible, in practice there will always be situations,

where no "suitable" modeling-construct is available. In such cases, the developer can either extend the meta-model (i.e. introduce a new modeling-construct) or he/she directly attaches implementation-level code.

The main purpose of the *ActitivyRefinement* stereotype is to allow a *MicroflowActivity* to refine a concrete *ActivityNode*. For example, a *MicroflowActivity* (`A1`) might contain an *ActivityNode* – with the name `N1` – to be refined. A second, *MicroflowActivity* (`A2`) might then use the tag `refinedNodes` to indicate, that it refines the node `N1` (from `A1`). As we will see in Section 5, this mechanism can not only be used to refine *MicroflowActivities* but also to integrate our meta-model with other meta-models.

3.2 Modeling Web Data Mashups

Based on the rather generic microflow meta-model introduced in the previous section, we will now present a model extension aiming to cover the most basic set of invocation activities related to Web mashups (i.e. "plain" HTTP and SOAP). Note, that the resulting model is far from "complete" and mainly tries to give the reader an idea of our meta-model's extension mechanism (see Section 4 for further details).

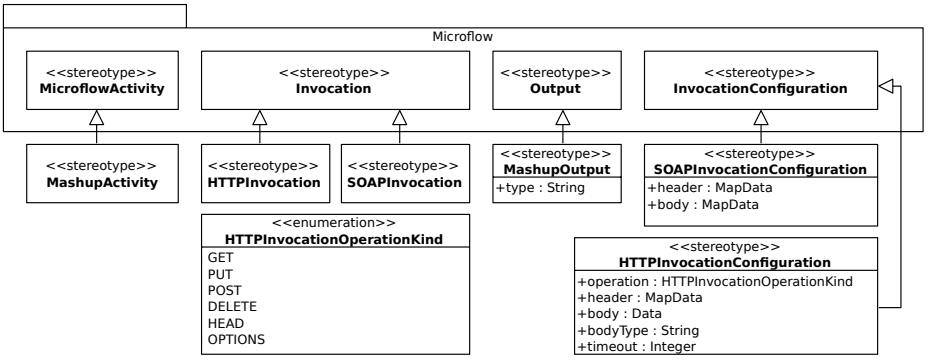

Fig. 3. The Mashup Meta-Model

Figure 3 illustrates the Mashup meta-model in its UML2 class diagram representation. *Invocation* is derived twice: *HTTPInvocation* and *SOAPInvocation*. The former is used to model a plain HTTP request (e.g. to retrieve a resource from a RESTful service or to post data to a JSON-based Web service). The stereotype *SOAPInvocation* indicates an invocation of a SOAP Web service. Finally, *MashupOutput* is derived from *Output*. The mandatory `type` tag is used to specify the MIME type of the data to be returned.

```
context HTTPInvocationConfiguration
  inv: self.operation->notEmpty()
  inv: self.operation = POST or self.operation = PUT
         implies self.body->notEmpty()
  inv: self.body->notEmpty() implies self.bodyType->notEmpty()
context HTTPInvocation
  inv: self.configuration.oclIsKindOf(HTTPInvocationConfiguration)
context SOAPInvocationConfiguration inv: self.body->notEmpty()
context SOAPInvocation
  inv: self.configuration.oclIsKindOf(SOAPInvocationConfiguration)
context MashupOutput inv: self.type->notEmpty()
```

Listing 2. OCL Constraints for the Mashup Model

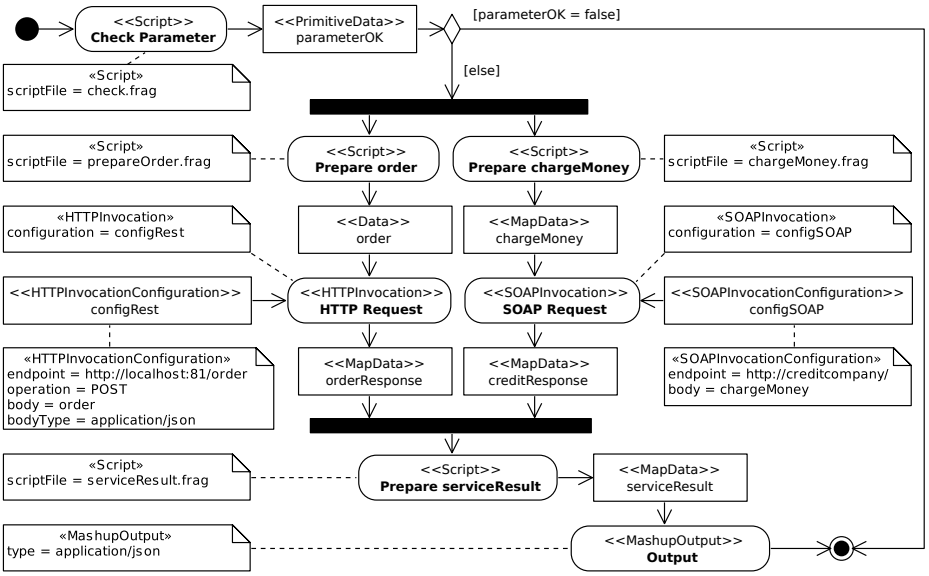

Fig. 4. Example Scenario

To give you an idea how a concrete instance of our Mashup meta-model might look like, let us consider a simple online shop. An HTML page resembles its user interface. Its backend is realized using a Web data mashup. Upon invocation, the it first has to place a new order in the internal inventory system of the company, which is reachable via a JSON-based Web service. Secondly, a billing request to the external SOAP Web service of an Credit card company is issued. Finally, the result of both invocations is passed back to the user interface (e.g. a simple HTML page). Figure 4 depicts the corresponding microflow model.

4 Exploring the Generalizability of the UML2 Profile

A generic and unified modeling approach implies, that – thanks to its generalizability – it is possible to accommodate models from similar approaches. This is achieved by mapping the model abstractions of one approach to the ones of the other. As this

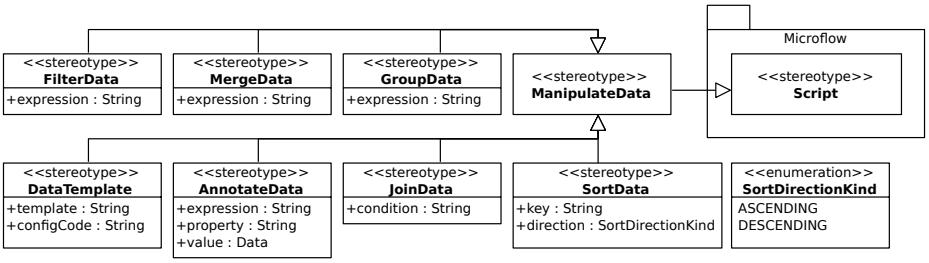

Fig. 5. EMML Extensions to the Mashup Meta-Model

is not always possible (e.g. because there is simply no matching modeling construct available), a generic modeling approach should provide an extension mechanism.

```
context ManipulateData
  inv: self.baseActivityNode.incoming->exists(in |
         Data.allInstances()->exists(data |in.source.oclIsTypeOf(ObjectNode)
           and in.source = data.baseObjectNode))
context AnnotateData
  inv: self.expression->notEmpty()
  inv: self.property->notEmpty()
  inv: self.value->notEmpty()
context FilterData inv: self.expression->notEmpty()
context GroupData inv: self.expression->notEmpty()
context JoinData
  inv: self.condition->notEmpty()
  inv: self.baseActivityNode.incoming->forAll(in |
         Data.allInstances()->select(data | in.source.oclIsTypeOf(ObjectNode)
           and in.source = data.baseObjectNode)->size() > 1
context MergeData
  inv: self.expression->notEmpty()
  inv: self.baseActivityNode.incoming->forAll(in |
         Data.allInstances()->select(data | in.source.oclIsTypeOf(ObjectNode)
           and in.source = data.baseObjectNode)->size() > 1
context SortData inv: self.key->notEmpty()
```

Listing 3. OCL Constraints for the EMML Mashup Model Extension

To explore the generalizability of our modeling approach we tried to map the concepts and model abstractions of the Enterprise Mashup Markup Language (EMML) [4]. EMML is an XML-based standard that supports the specification of processing flows for Web mashups in a platform- and vendor-independent manner. Table 1 contains a list of EMML statements (taken from the reference [4]) and shows how each statement can be mapped to our UML2 profile. In Table 1a we can see, that many statements (e.g. control flow-related) can directly be mapped to plain UML2 (e.g. `<if>`).

For a large part of the domain-specific statements (e.g. `<mashup>`) this is also the case. To cover the remaining, we had to extend our model. Figure 5 shows, that we have extended the *Script* stereotype – the primary extension point of our model – to introduce 8 new stereotypes. Listing 3 shows the corresponding OCL constraints (e.g. *JoinData* needs at least two incoming activity edges originating *Data* objects) and Table 1b shows how they are mapped to EMML. The remaining statements are listed in Table 1c. We considered them either to be "generic" in a sense that they are not very specific for the

Table 1. Mapping of EMML language elements to UML2

(a) Plain UML2

EMML	UML2
<input>	ActivityParameterNode
<variables>	ObjectNode / ObjectFlow
<include> <macro> <macros>	Activity
<if> <for> <foreach> <break> <while>	DecisionNode
<parallel>	ForkNode / JoinNode
<sequence>	ControlFlow

(b) UML2 Stereotypes

EMML	UML2
<mashup>	*MashupActivity*
<directinvoke> <invoke>	*Invocation*
<annotate>	*AnnotateData*
<filter>	*FilterData*
<group>	*GroupData*
<join>	*JoinData*
<merge>	*MergeData*
<sort>	*SortData*
<xslt>	*DataTemplate*
<output>	*Output*

(c) Script Fallbacks

EMML	UML2
<script> <select> <appendresult> <assert> <assign> <constructor> <template> <display> <datasource> <sql*>	*Script*

domain of "data mashups" (e.g. `<constructor>`) or to mainly exist for debugging purpose (e.g. `<assert>`). Hence, we used the *Script* "fallback" to cover them.

As we could show, our modeling approach provides a model-driven abstraction that can be used to model the essence of mashups expressed in languages like EMML in a technology-independent way that supports implementing features for model-driven generation of system integration code, analysis, or adaptation based on the abstract models. EMML code could be generated from our models and Section 7 will show that it is feasible to implement a model-driven interpreter that can execute instances of our meta-model on-the-fly.

5 Integration of the UML2 Profile with Existing Models

Different meta-models can be integrated via a common meta-meta-model, like MOF for UML2. That is, every single meta-model to be integrated has to be defined using the same meta-meta-model. The profile definition mechanism of UML2 provides straightforward means to define meta-models. As a standard modeling language, lots of different UML2 profiles and UML2-derived meta-models have been proposed. Hence, basing model integration on the common UML2 meta-model allows for an straightforward integration with other UML2-based meta-models.

Using an extension of our illustrative example, we will demonstrate the model integration capabilities of our mashup meta-model via the standard UML2 extension mechanisms. As mentioned before, mashups may very likely be used in larger architectures. For instance, our example mashup from Section 3.2 may be used by a macroflow [13], a long-running, interruptible process flow which depicts the business-oriented process perspective (e.g. a business process).

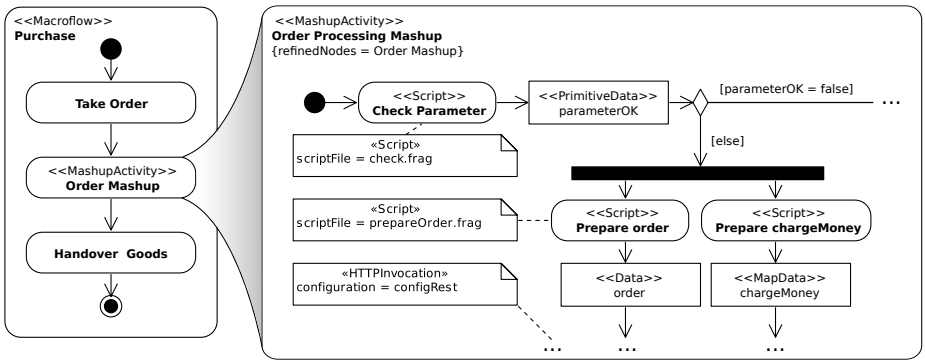

Fig. 6. Integrating the Mashup Model with a Macroflow Model

Suppose that the company from our example scenario (see Section 4) also provides a physical "brick and mortar" store. The left side of Figure 6 depicts a very simplistic and high-level macroflow model of the whole buying process. The first as well as the last activities have to be conducted by a human (i.e. the shop assistant). That is, after taking the order, the original mashup model (from Section 4) shall be used to process it. Hence,

we insert an activity node (*Order Mashup*) to be refined. Using the *ActivityRefinement* stereotype and the `refinedNodes` tag, we can then specify that our order processing mashup refines the mentioned activity node in the macroflow model.

This way of integrating different compatible meta-models using a tagged value introduced in the mashup profile (i.e., `refinedNodes`) is one way of model integration – in this case with other activity models. Other types of UML2 models can easily be integrated in the same way. Another option is named-based matching. For instance, the object nodes in our mashup models can easily be matched by name with the corresponding classifiers in class or component models that describe them in detail. Class models can also be used to describe the service interfaces used in a mashup.

A model-driven generator or interpreter can then use the linking tagged values or names to navigate both models and generate code for different system artifacts. The big benefit of our UML2 profile is that mashups can easily be integrated with models in other types of models and that UML2 already provides a wide variety of models that can be used to describe all kinds of other artifacts relevant for mashups.

6 Implementing a Model-Driven Tool Chain

The presented UML2 meta-models have been developed and specified using a textual DSL. Frag [14, 15], a tailorable language, specifically designed for the task of defining DSLs, provides the syntactic foundation of this DSL. Among other things, Frag supports the tailoring of its object system and the extension with new language elements. Hence, it provides a good basis for defining a UML2-based textual DSL because it is easy to tailor Frag to support the definition of the UML2 meta-classes. Frag automatically provides us with a syntax for defining application models using the UML2 meta-classes. In addition Frag also provides a constraint language similar to OCL as well as a model validator. Using the model validator we can easily check a models conformance to its meta-models as well as its model constraints.

```
# define a new stereotype
FMF::Stereotype create MashupOutput \
  -superclasses Microflow::Output \
  -attributes { type String }
# define a new model constraint
MashupOutput addInvariant [notEmpty [self type]]
```

Listing 4. Frag DSL example

Note, that the textual syntax of the DSL is mainly intended to be used internally in the model validator, as a common syntax for model integration, and for debugging purposes. The developers should mainly work with UML2 and OCL tools to define the models and constraints. The main contribution of our prototypical tool chain is to validate and demonstrate that a model validation support following our concepts is feasible and can be implemented with moderate effort from scratch.

7 Implementing a Model-Driven Interpreter

As a proof-of-concept, we have also implemented a basic model-driven interpreter, that is able to execute instances of our mashup meta-model on-the-fly. Using the Frag language, and mainly due to its realization of the transitive mixins concept [15], it could

be implemented in roughly 450 lines of code. Mixins allow us (among other things) to add methods to classes dynamically at runtime.

```
# create executor classes
FMF::Class create MashupActivityExecutor -method execute args { ... }
FMF::Class create ScriptExecutor -method execute args { ... }
# add mixins
Mashup::MashupActivity mixins MashupActivityExecutor
Microflow::Script mixins ScriptExecutor
```

Listing 5. Defining Mixin Classes

Thus, the basic idea of our model execution-approach is to use mixins to extend our (Frag-specified) meta-model with additional execution functionality. For instance, Listing 5 shows how we define two mixin classes (`MashupActivityExecutor` and `ScriptExecutor`), both implementing the method `execute`. For every defined stereotype a corresponding executor mixin – containing the execution-logic – is needed. For instance, the execution-logic of the `MashupActivityExecutor` is to execute the model's initial node. The initial node's execution logic is to traverse its outgoing activity edge and execute the next activity node. In Listing 5 we can see, that the previously defined mixin classes are then directly attached to the classes of the meta-model (e.g. `Microflow::Script`).

```
# define a model instance
UML2::Activity create A1
Mashup::MashupActivity create M1 -baseActivity A1
# execute the model instance
M1 execute
```

Listing 6. Executing a *MashupActivity*

Having the mixin classes attached, it is then possible to directly execute any instance of our meta-model. Listing 6 depicts both the instantiation of the meta-model as well as the execution of the newly created instance via the `execute` method.

8 Related Work

A considerable amount of work has been done on the design and development of DSLs that are tailored specifically to facilitate the development of Web mashups (see e.g. [1–3]). In particular, the idea of seeing Web mashups as compositions of Web services and Web data leads to the design of numerous service composition languages. For instance, the Bite language [7] has been proposed as a simplified variant of the Web Services Business Process Execution Language (WS-BPEL) [16], a current standard technology for business process execution in the context of Web services. Like our approach this approach uses a behavioral model as the foundation of a mashup model. But BPEL is designed for long-running, transactional business processes (macroflows) and contains many language elements not useful for mashup composition, whereas our approach offers a model focused on the short-running microflows typically required for mashup composition tasks. Rosenberg et al. [8] demonstrate the applicability of Bite to model RESTful Web services and collaborative workflows.

Our model-based approach does not compete with the already existing languages and approaches. But rather it provides a model-driven abstraction that can be used to model the essence of mashups expressed in these languages. This has been demonstrated in Section 4 for the Enterprise Mashup Markup Language [4], a standard proposed by

the Open Mashup Alliance. In contrast to our approach, the existing modeling approaches are not based on a standard modeling language that provides convenient ways to model other system parts as well like the UML2 (e.g. in UML2 service interfaces can be modeled as extensions of UML2 class diagrams). Our approach can be used to augment those other mashup modeling languages with links to UML2 models for other system parts via the standard UML2 extension mechanisms.

Model-driven development in the context of Web mashups and Web data integration is nothing new and numerous approaches have been presented before. For example, Daniel et al. present mashArt [5], a model-driven approach to UI and service composition on the Web, consisting of component model for mashup components as well as an event- and flow-based service composition model. A meta-model for context-aware component-based mashup applications is presented by Pietschmann et al. [6]. The model provides means to describe all necessary application aspects on a platform-independent level, such as its components, control and data flow, layout, as well as context-aware behavior. Koch et al. present UWE [17], a model-driven approach for Web application development. The proposed UML2 profile aims to cover the entire development life cycle of Web systems and therefore clearly surpasses the scope of our own meta-model. Similarly, Kapitsaki et al. [18] also suggest a UML2 profile for modeling Web applications using UML2 class and state transition diagrams. A conceptual modeling approach to business service mashup development is presented in [19]. Bozzon et al. demonstrate the feasibility of modeling Web mashups as Business Processes using BPMN (Business Process Management Notation). In summary, these approaches attach great importance to the integration of the data and the user interface layer – which is the main focus of the meta-models of these approaches.

In contrast to these approaches, our approach tries to be as generic as possible and focus on the microflow abstraction needed to support features for model-driven generation of system integration code, analysis, or adaptation. Thus, our meta-model constitutes the bare minimum needed to model the microflows of Web mashups. Also, our main focus lies in the Web data integration and service composition aspect of Web mashups. In future extension of our model we plan to extend it to also support the user interface layer integration.

9 Conclusion and Future Work

In this paper we introduced an UML2 profile for semi-formally modeling the essence of Web data mashups based on activity diagrams and formal constraints in the OCL. We divided our meta-model into an abstract microflow layer and a mashup specific layer. We were able to show the applicability of our approach in a prototype implementation, realizing a mashup DSL and a model-driven interpreter. We showed the generalizability of our approach by mapping it to a standard mashup language, the EMML. We argued and showed how other UML2 diagrams can be integrated with our approach. Hence, the UML2 profile together with the model-driven approach help to make the mashup approach usable in a system integration context, in which the mashups and other dependent components must be changed together. The approach can potentially be used to better support the adaptation and analysis of mashups – especially together with other system components. As future work we plan to apply our approach in for these tasks.

References

1. Maximilien, E.M., Ranabahu, A., Gomadam, K.: An Online Platform for Web APIs and Service Mashups. IEEE Internet Computing 12(5), 32–43 (2008)
2. Vallejos, J., Huang, J., Costanza, P., De Meuter, W., D'Hondt, T.: A programming language approach for context-aware mashups. In: Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups, Mashups 2009/2010, pp. 4:1–4:5. ACM, New York (2010)
3. Sabbouh, M., Higginson, J., Semy, S., Gagne, D.: Web mashup scripting language. In: Proceedings of the 16th International Conference on World Wide Web, WWW 2007, pp. 1305–1306. ACM, New York (2007)
4. Open Mashup Alliance: Enterprise Mashup Markup Language, `http://www.openmashup.org/omadocs/v1.0/`
5. Daniel, F., Casati, F., Benatallah, B., Shan, M.-C.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 428–443. Springer, Heidelberg (2009)
6. Pietschmann, S., Tietz, V., Reimann, J., Liebing, C., Pohle, M., Meißner, K.: A metamodel for context-aware component-based mashup applications. In: Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2010, pp. 413–420. ACM, New York (2010)
7. Curbera, F., Duftler, M., Khalaf, R., Lovell, D.: Bite: Workflow Composition for the Web. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 94–106. Springer, Heidelberg (2007)
8. Rosenberg, F., Curbera, F., Duftler, M.J., Khalaf, R.: Composing RESTful Services and Collaborative Workflows: A Lightweight Approach. IEEE Internet Computing 12(5), 24–31 (2008)
9. Mellor, S.J., Clark, A.N., Futagami, T.: Guest Editors' Introduction: Model-Driven Development. IEEE Software 20, 14–18 (2003)
10. Bock, C.: Unified Behavior Models. Journal of OO-Programming 12(5), 65–68 (1999)
11. Aguilar-Savén, R.S.: Business process modelling: Review and framework. International Journal of Production Economics 90(2), 129–149 (2004)
12. Object Management Group: UML 2.4.1 Superstructure, `http://www.omg.org/spec/UML/2.4.1`
13. Hentrich, C., Zdun, U.: Process-Driven SOA - Proven Patterns for Business-IT Alignment. CRC Press, Taylor and Francis, Boca Raton (2012)
14. Zdun, U.: Frag, `http://frag.sf.net/`
15. Zdun, U.: Tailorable language for behavioral composition and configuration of software components. Comput. Lang. Syst. Struct. 32(1), 56–82 (2006)
16. OASIS: Web Services Business Process Execution Language, `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`
17. Koch, N., Knapp, A., Zhang, G., Baumeister, H.: Uml-Based Web Engineering. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modelling and Implementing Web Applications. Human–Computer Interaction Series, pp. 157–191. Springer, London (2008)
18. Kapitsaki, G.M., Kateros, D.A., Pappas, C.A., Tselikas, N.D., Venieris, I.S.: Model-driven development of composite web applications. In: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2008, pp. 399–402. ACM, New York (2008)
19. Bozzon, A., Brambilla, M., Facca, F.M., Carughu, G.T.: A Conceptual Modeling Approach to Business Service Mashup Development. In: Proceedings of the 2009 IEEE International Conference on Web Services, ICWS 2009, pp. 751–758. IEEE Computer Society, Washington, DC (2009)