

# FlexMash – Flexible Data Mashups Based on Pattern-Based Model Transformation

Pascal Hirmer<sup>(✉)</sup> and Bernhard Mitschang

Institute of Parallel and Distributed Systems, University of Stuttgart,  
Universitätsstr. 38, 70569 Stuttgart, Germany  
[pascal.hirmer@ipvs.uni-stuttgart.de](mailto:pascal.hirmer@ipvs.uni-stuttgart.de)  
<http://www.ipvs.uni-stuttgart.de>

**Abstract.** Today, the ad-hoc processing and integration of data is an important issue due to fast growing IT systems and an increased connectivity of the corresponding data sources. The overall goal is deriving high-level information based on a huge amount of low-level data. However, an increasing amount of data leads to high complexity and many technical challenges. Especially non-IT expert users are overburdened with highly complex solutions such as Extract-Transform-Load processes. To tackle these issues, we need a means to abstract from technical details and provide a flexible execution of data processing and integration scenarios. In this paper, we present an approach for modeling and pattern-based execution of data mashups based on Mashup Plans, a domain-specific mashup model that has been introduced in previous work. This non-executable model can be mapped onto different executable ones depending on the use case scenario. The concepts introduced in this paper were presented during the Rapid Mashup Challenge at the International Conference on Web Engineering 2015. This paper presents our approach, the scenario that was implemented for this challenge, as well as the encountered issues during its preparation.

**Keywords:** ICWE rapid mashup challenge 2015 · Data mashups · Transformation patterns · TOSCA · Cloud computing

## 1 Context and Goals

Nowadays, the complexity and size of the IT systems used in enterprises constantly increase. Especially in the area of data processing and integration, this leads to high costs as well as communication effort between domain-specific users, e.g., business persons, and IT experts that implement the data processing. This oftentimes results in hand-made, monolithic, non-flexible solutions that are exclusively suitable for a few number of use cases. For example, Extract-Transform-Load (ETL) process models and their execution can usually only be used for specific scenarios, i.e., they offer nearly no flexibility. Furthermore, existing data mashup or data streaming solutions mostly offer a single possibility how data is processed, fulfilling only a limited number of user requirements.

To tackle these issues, we need a data mashup solution that offers domain-specific modeling as well as a corresponding technical execution of data processing and integration depending on the use case scenario. That is, its execution has to flexibly suite a scenario’s special requirements, e.g., robustness, scalability, security, efficiency. Using a non-technical, domain-specific model enables users to define data processing and integration scenarios they are interested in without any need of implementation and execution details. Aside the domain-specific model, the user should have the possibility to define requirements that are fulfilled by the mashup execution. In previous work, we introduced *Mashup Plans* [11], a graph-based model that enables domain-specific modeling of data mashups. Mashup Plans enable modeling data sources as so called *business objects* [6, 14] that represent domain-specific objects, e.g., an enterprise information system or a production machine, and abstract from low-level data structures such as databases, ontologies, sensors or unstructured text. In the context of this paper as well as of previous work, these business objects are called *Data Source Descriptions (DSD)*. Furthermore, Mashup Plans contain *Data Processing Descriptions (DPD)* that abstract from fine-grained data operations and offer generic, easy-to-use, domain-specific data processing operations (e.g., filter or combine) that can be mapped onto a multitude of implementations that depend on the context the DPD is used in. Mashup Plans are modeled as shown in Fig. 1. In this paper, we introduce a new approach to transform Mashup Plans into alternative, executable formats depending on the requirements set by the use case scenario. Note that due to the implementation focus of this paper, we are not looking into conceptual details.

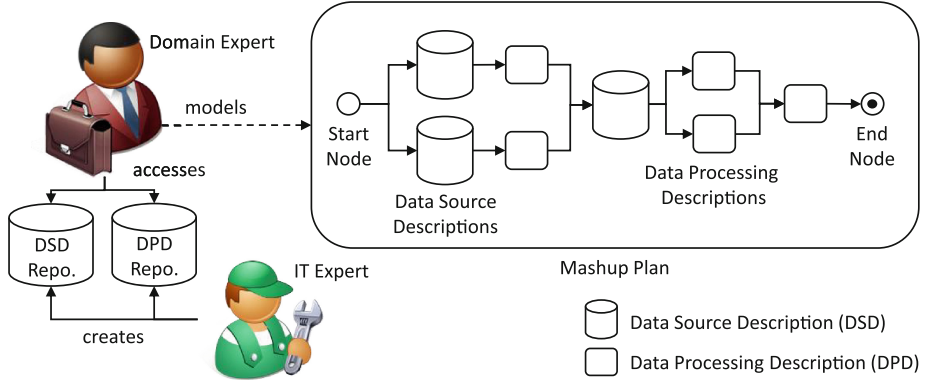
The remainder of this paper is structured as follows: in Sect. 2, we describe basic concepts that are necessary to explain our approach. Section 3 describes related work. In Sect. 4, the main contribution of our paper is presented: we introduce an approach for pattern-based Mashup Plan transformation and execution. Section 5 describes the maturity of our tool and Sect. 6 its features. After that, in Sect. 7, the prototypical implementation of our approach is presented, in Sect. 8, the presented demo flow is shown, in Sect. 9 we describe the challenge preparation and the results are subsequently evaluated in Sect. 10. Finally, Sect. 11 summarizes the results and gives an outlook to future work.

## 2 Basic Concepts

In this section, we describe basic concepts that are necessary to comprehend the approach presented in this paper. These concepts are (i) Mashup Plans, as introduced in previous work, and (ii) the Topology and Orchestration Specification for Cloud Applications (TOSCA) that is used for provisioning of the mashup’s execution components in a cloud computing environment.

### 2.1 Mashup Plans

A Mashup Plan is a non-executable, domain-specific model to define data mashup (i.e., ad-hoc data processing and integration) scenarios and was



**Fig. 1.** Overview of Mashup Plans (based on [11])

introduced in our previous work [11]. The modeler of Mashup Plans is usually a domain-expert such as a business person, without any knowledge of technical details. As depicted in Fig. 1, a Mashup Plan is a cohesive, directed flow graph based on the *Pipes and Filter* pattern [15] containing two types of nodes and a single edge type representing the data and control flow between these nodes. The node types are subdivided into Data Source Descriptions (DSD) and Data Processing Descriptions (DPDs) as well as a single start and end node. Data Source Descriptions offer a non-technical way to model data sources, without having to know about low-level details such as data base ports, URLs, etc. These DSDs are based on so called business objects as described by [6, 14]. The second type of nodes, the DPDs, describe how the data is processed, i.e., how it is filtered, aggregated, analyzed, or otherwise modified. That is, a DPD describes an operation, i.e., a piece of code, that processes the data. The actual implementation of the DPD depends on its context. As a consequence, different implementations exist for a single DPD, depending on the data types, data structures, etc. The mapping from DSDs and DPDs to their corresponding implementation can be realized by a rule-based approach as described by Reimann et al. [19]. When modeling Mashup Plans, the following restrictions have to be considered: (i) a Mashup Plan contains a single start node to indicate the entry point of the flow, (ii) a completely modeled Mashup Plan contains at least one Data Source Description and at least one Data Processing Description, and (iii) a Mashup Plan contains a single output represented by an end node (depicted in Fig. 1). The technical properties of DSDs and DPDs that are used to model Mashup Plans are defined once by IT experts who store them in corresponding repositories. This enables Mashup Plan modeling by domain-experts based on the DSD and DPD repositories without them having to specify any technical details. A concrete example of a modeled Mashup Plan using the FlexMash data mashup tool is depicted in Fig. 5(a).

Note that we slightly modified the concept of Mashup Plans in this paper in contrast to previous work by adding the start node. In previous work, the entry

point of Mashup Plans was defined through the data source descriptions. As a consequence, DSDs could not contain incoming connections and represented the starting points of the flow. However, in some use cases it is necessary to integrate data sources not only in the beginning but also within the flow (e.g., the Twitter data source in Fig. 5(a)). As a consequence, we added the start node to the Mashup Plan.

## 2.2 TOSCA

In this section, basic concepts of the Topology and Orchestration Specification for Cloud Applications (TOSCA) are introduced that are necessary to comprehend the approach presented in this paper. The following section is based on [10].

TOSCA is a standard of OASIS to describe cloud applications in a portable way. TOSCA-based descriptions define (i) the structure as well as (ii) the management functionalities of cloud-based applications. Although TOSCA is a relatively new standard, several tools exist that ease modeling, provisioning, and management of TOSCA-based applications. The open source ecosystem *OpenTOSCA*, for example, includes a graphical modeling tool called *Winery* [13] and a plan-based provisioning and management runtime environment [3], which can be used to provision and manage TOSCA applications fully automatically. Further details on the TOSCA standard can be found in the official OASIS TOSCA specification [16], TOSCA Primer [17], or Binz et al. [2].

The core of the application description in TOSCA is the *Topology Template*, a directed graph containing *Node Templates* (vertices) and *Relationship Templates* (edges). Node Templates may describe all components of an application, including all software and hardware components. The relations between those Node Templates are represented by Relationship Templates. Node and Relationship Templates are typed by *Node Types* and *Relationship Types*, respectively. Types define the semantics of the templates, as well as their properties, provided management operations, and so on. Types can be refined or extended by an inheritance mechanism. *TOSCA Policies* are used to define non-functional requirements for the provisioning of an application. Using TOSCA Policies, it is possible to determine costs, security, availability, scalability or similar non-functional requirements. TOSCA specifies an exchange format called *Cloud Service Archive* (CSAR) to package Topology Templates, types, associated artifacts, plans, and all required files into one self-contained package. This package is portable across different standards-compliant TOSCA runtime environments [4].

## 3 Related Work

In the past, many data mashup solutions have been introduced in science and industry that are built to enable an ad-hoc processing and integration of data. Usually these solutions offer a graphical modeling tool that enables the users to define data sources and data operations as well as the way data is processed.

Well-known examples are *Yahoo! Pipes*<sup>1</sup>, *Intel MashMaker*<sup>2</sup> and the *IBM Infosphere MashupHub*<sup>3</sup>. These enterprise-ready solutions offer a lot of functionality in regard to the data sources they are able to integrate as well as the data processing operations they support. However, existing solutions only offer a single possibility for execution, i.e., they have a single, static implementation. Nowadays, the user requirements may differ significantly, especially when it comes to data processing and integration. In production environments, for example, it is very important that data is processed in *real-time*, i.e., a very efficient execution has to be supported. Furthermore, in businesses, for example, robustness and security are very important aspects. In other scenarios, the coping with huge amounts of data has to be supported, and so on. However, current mashup solutions cannot cope with these heterogeneous requirements. In this paper, we tackle this issue by introducing a flexible data mashup execution based on user requirements.

Furthermore, existing approaches define abstract, non-technical models to describe data processing and integration scenarios similar to the introduced Mashup Plans. However, oftentimes these approaches do not offer a sufficient abstraction from technical details. For example, many modeling nodes in *Yahoo! Pipes* require the knowledge of programming concepts such as string builders, regular expressions, HTML scraping, *for each*-loops, and so on. This limits the usage to software developers that have to know about technical details. Using so called *business objects* [6, 14], we can enable modeling data mashup scenarios based on the user's domain. This further enables a widely usable data mashup solution, e.g., for business users and for technical experts as well.

In previous work [11], we already introduced the modeling of Mashup Plans as well as some basic ideas of their transformation. These Mashup Plans can be modeled using a variety of different formats, e.g., also using established standards such as BPMN, XML or JSON. All these abstract languages have to be further transformed onto an executable level as described in Sect. 4.2. As a consequence, Mashup Plans offer a generic means to define data mashup scenarios and are not bound to a specific format. In this paper, we focus on the transformation of this model to an executable representation based on patterns, and we describe how these concepts were applied during the ICWE Mashup Challenge 2015.

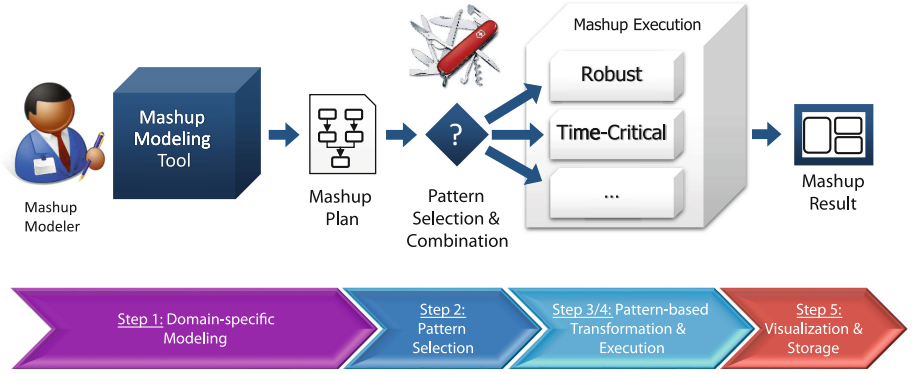
## 4 Flexible Data Mashups Based on Pattern-Based Model Transformation

This section describes our proposed mashup approach: flexible data mashups based on pattern-based model transformation. We subdivide the approach into five main steps as depicted in Fig. 2: (i) the modeling of Mashup Plans, (ii) the selection of transformation patterns, (iii) the pattern-based transformation of Mashup Plans into an executable format, (iv) the cloud-based data mashup execution based on user requirements, and (v) the storage and/or visualization

<sup>1</sup> <https://pipes.yahoo.com/pipes/>.

<sup>2</sup> <http://intel.ly/1BW2crD>.

<sup>3</sup> <http://ibm.co/1Ghvx27>.



**Fig. 2.** Overall approach of flexible data mashups (based on [11])

of the derived result. These overall steps are based on previous work [11]. Note that in the following, the terms *pattern* and *transformation pattern* are used synonymously.


After modeling of the Mashup Plan that defines the data as well as how it is processed and integrated, the user can select patterns that represent his/her requirements for the mashup execution. That is, each pattern can fulfill certain user requirements, such as efficiency or robustness. The transformation of the domain-specific, non-executable model to an executable representation is done based on the selected patterns. Finally, the mashup is executed in a suitable engine. The result of the execution can be stored or visualized.

The modeling of Mashup Plans (Step 1) has already been described in previous work, as a consequence, the Mashup Plan modeling step depicted in Fig. 2 will not be described here. Its description can be found in [11]. Furthermore, the use of the mashup result (Step 5) is mostly application-dependent and therefore out of scope of this paper.

#### 4.1 Step 2: Transformation Pattern Selection

In this section, we introduce the transformation patterns, how they are selected and how they can be parameterized. Patterns are high-level descriptions of established practices to solve reoccurring problems. Each pattern can be implemented in a different fashion, i.e., patterns offer an abstract solution to specific problems and can be mapped onto corresponding solution implementations [8]. Each of the *transformation patterns* introduced in this paper fulfills certain user requirements for the data mashup execution.

During modeling of the Mashup Plan, the modeler has the possibility to select the patterns him/herself based on his or her requirements. In case the Mashup Plan modeler is a business person without any knowledge of the specific technical requirements, this decision can also be made by (IT) experts supporting the modeler by analyzing the mashup scenario. To be able to select a pattern,

	Transformation Pattern: Robust Mashup
<u>Problem:</u>	Robustness is an <b>important factor</b> for IT systems, especially in <b>enterprise applications</b> and systems. It stands for many factors such as <b>stability, error tolerance, logging</b> etc. that have to be fulfilled in a robust environment.
<u>Solution:</u>	A <b>robust execution engine</b> that supports error handling, logging as well as data persistence is used.
<u>Example:</u>	An exemplary pattern implementation could, e.g., be realized using a <b>workflow engine</b> such as Apache ODE, the Oracle workflow engine or the WSO2 engine using BPEL as execution language. These engines <b>provide all the necessary factors</b> to ensure robustness.
<u>Evaluation:</u>	The Robust Mashup pattern can be used in enterprise environments in which, e.g., workflows are already established. By using this pattern, <b>robustness can be guaranteed</b> which is the most important factor in enterprises. However, of course there are some <b>setbacks regarding runtime efficiency</b> .
<u>Combination:</u>	<a href="#">Secure Mashup Pattern</a> ; <a href="#">Big Data Mashup Pattern</a> ;

**Fig. 3.** Example of a pattern catalog entry – Robust Mashup

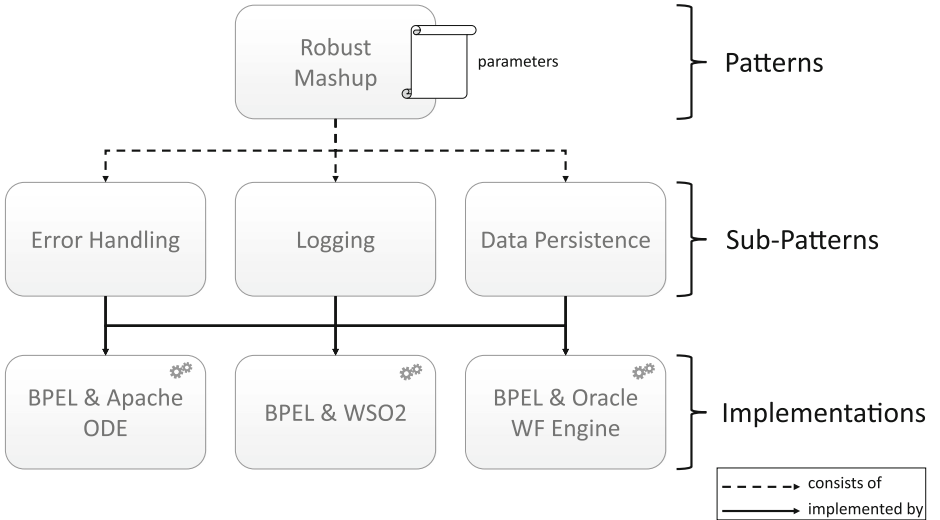
the modeler has to know about all existing patterns, know about their abilities as well as their limitations. Furthermore, the modeler has to know how these patterns can be combined in a reasonable manner. To enable this, we created an extendable pattern catalog that describes widely used patterns regarding data mashup processing. Each entry in this catalog describes a single transformation pattern and has the following content: (i) a description containing the **problem** that is solved by the pattern, (ii) the **solution** the pattern offers to solve this problem, (iii) an **example** how the pattern can be applied, (iv) a short **evaluation**, and (v) information about if and how it can be **combined** with other transformation patterns. When selecting a pattern, the user usually has to define additional parameters that are necessary to find a corresponding implementation. For example, when selecting the time-critical mashup pattern, the user has to specify the maximal time the execution may take. When selecting the robust mashup pattern, the user e.g., has to specify whether error handling is needed, logging has to be supported, etc. This parameterization is done during the pattern's selection. An exemplary entry of the pattern catalog is shown in Fig. 3. As depicted, it contains a textual description of how the pattern can be applied. The selected patterns influence the manner the mashup is executed, e.g., when the depicted pattern *Robust Mashup* is selected from the pattern catalog, the mashup is executed in a robust manner, e.g., using a workflow engine. That is, the selected patterns give a directive for the executable format the mashup plan is transformed into and, as a consequence, for the execution components.

## 4.2 Step 3: Pattern-Based Transformation

In this section, we describe how the non-executable Mashup Plan is transformed into an executable representation based on patterns. The executable format the

Mashup Plan is transformed into, depends on the patterns that were selected in the previous step. The mapping of the Mashup Plan onto the executable model as well as the selection of the execution engine that is being used to execute it, is chosen using a rule-based transformation approach, similar to the one described by [18, 19]. To structure patterns and connect them to corresponding implementations, we use so called *pattern graphs*. A pattern graph is a tree-based, directed graph containing nodes and edges. A node in the pattern graph represents either a pattern or an implementation. An edge from one node to another represents a specialization. There are two types of edges. The “*consists of*” edge is used to connect patterns and indicates that a pattern consists of several sub-patterns. As a consequence, the problem described by the pattern can only be solved, if all of its sub-patterns are applied. The second edge type “*implemented by*” is used to connect to the implementation nodes. If a pattern is connected to one or more implementations, it means that it can be realized by either one of them. In this case, one of them has to be selected either manually or automatically.

To summarize, a generic pattern at the root node of the tree is getting more and more concrete by being subdivided into sub-patterns and finally into implementation fragments. As a consequence, patterns can be structured hierarchically through different abstraction levels. Furthermore, a single pattern can be realized by different implementations. The root node of the pattern graph represents the most abstract pattern, which is the pattern described textually in the pattern catalog. That is, a different pattern graph exists for each entry of the pattern catalog. Which path in the pattern graph is chosen in order to reach the implementation at the leaf nodes, depends on the pattern’s parameterization. This decision is made based on rules that are evaluated during traversal of the



**Fig. 4.** Pattern graph example

pattern graph. These rules compare the parameters of the pattern with predefined properties of the implementations to find the most suitable one. Note that our approach will always find an implementation, however, it is not guaranteed that it can fulfill all given user requirements. In this case, the user has to decide whether to use the selected implementation or not. An exemplary pattern graph for the *Robust Mashup* pattern is depicted in Fig. 4.

Note that for a single selected pattern, this rule-based transformation approach can be applied in a straight-forward manner. However, if several patterns are combined, the determination of a suitable pattern implementation is much more complex and is currently part of our ongoing work.

Once a suitable implementation is found, the transformation of the Mashup Plan to a suitable executable representation can be processed. We use predefined, modularized implementation fragments that are scripted together to create the executable model. For example, if the execution is done using a workflow engine, we can create the executable workflow automatically using, e.g., invoke nodes of the Business Process Execution Language (BPEL) to execute the operations defined in the Mashup Plan. The programming logic of the DSDs and DPDs is stored in code fragments, e.g. Java Web Services that are executed by the workflow. In other examples, e.g., when using the Node-RED<sup>4</sup> execution engine, the transformation works in a similar fashion by connecting predefined, JavaScript code fragments. The implemented pattern transformations for the ICWE Rapid Mashup Challenge are described in Sect. 9.

### 4.3 Step 4: TOSCA-Based Deployment and Execution

To execute our data mashups, several software components are necessary that process the data flow, provide the programming logic of DSDs and DPDs and visualize or store the result, e.g., in a database or data warehouse (Step 5). Our goal is provisioning these components *on-demand*, i.e., only if a data mashup is initiated. The provisioning of these components is done once on the first execution of the mashup flow. If the mashup is not needed anymore, the components can be shut down to save costs. To enable this, we use approved cloud computing technologies, the OASIS standard TOSCA [16], as well as the results of our previous work [10]. In the first step, the components to be provisioned are received by traversing the pattern graph. Based on this information, a TOSCA topology can be created automatically containing the necessary components as well as information about how they are connected. This can be achieved by the concept of Node Templates and Relationship Templates provided by the TOSCA standard. In our previous work [10], we introduced an approach for automated completion of topologies for TOSCA-based cloud applications, which enables completing topologies that are only containing application-specific components and are missing platform as well as infrastructure components. Using these concepts, we can automatically complete the topology and create a TOSCA cloud service archive (CSAR), a self-contained package, containing all the information

<sup>4</sup> <http://nodered.org/>.

and software components necessary to provision applications in a cloud environment. For example, if the implementation contains a workflow engine to be provisioned, the necessary components to run it, such as a web server, an operating system and an instance of a cloud provider are added automatically to the topology and, as a consequence, to the CSAR. Using the plan generator extension of our TOSCA runtime OpenTOSCA [4], this cloud service archive can be used for an automated deployment of the components in the cloud. The automated deployment and execution of the mashup can be initiated using *management plans*, as described by [5]. The interested reader is referred to [2, 3, 10] and [16] for more information about TOSCA and the OpenTOSCA ecosystem. The implementation of these concepts is part of our ongoing work.

## 5 FlexMash – Level of Maturity

This section describes the current maturity of the implementation of our previously described approach. We implemented a prototype of FlexMash and used it in two different use case scenarios besides the one for the ICWE Rapid Mashup Challenge described in Sect. 7. In the first scenario, sensor data is integrated and processed to determine high-level situations in smart environments. For this implementation, we developed our prototype to support the stream-based processing of sensor data. The detailed results are described in [9]. The second use case implements a data mashup for exception escalation in advanced manufacturing environments, which is described in [12]. In this use case, exceptions in manufacturing environments are recognized and analyzed based on different data sources. The executed data mashup provides a result to find and solve occurred problems in an efficient manner by processing and integrating the corresponding data of the sources.

The current version of our prototype is tailored to these use cases. However, it offers a high degree of extensibility, which enables an easy adding of different data sources, data operations, patterns, execution formats and engines.

## 6 FlexMash – Feature Checklist

In this section, the features of the current state of FlexMash’s implementation are described based on the ICWE Rapid Mashup Challenge checklist, which contains information about important properties and design choices of mashup tools to enable their categorization. The feature checklist is based on related work and is subdivided into two parts: (i) an overall *mashup feature checklist* as described in [7] (Chap. 6), and (ii) a *mashup tool feature checklist* as described in [1]. The detailed information about the single entries are provided in these references.

- Mashup Feature Checklist
  - **Mashup Type:** Data mashups
  - **Component Type:** Data components
  - **Runtime Location:** Both client and server

- **Integration Logic:** Orchestrated integration
- **Instantiation Lifecycle:** Stateless
- Mashup Tool Feature Checklist
  - **Targeted End-User:** Non programmers
  - **Automation Degree:** Semi-automation
  - **Liveness Level:** Level 3 – Automatic compilation and deployment, requires re-initialization
  - **Interaction Technique:** Visual language (Iconic)
  - **Online User Community:** None (yet)

## 7 ICWE Rapid Mashup Challenge – Mashup Scenario

In this section, we introduce the scenario that was implemented and presented during the ICWE Rapid Mashup Challenge 2015. For this challenge, we introduced a specific mashup scenario according to the requirements. The goal of the challenge was integrating specific data sources in an “*elegant*” manner using new mashup tools and approaches. The choices of the data sources were as follows: (i) the New York Times API<sup>5</sup>, which can be used to receive articles and other news items, (ii) the YouTube API<sup>6</sup> to integrate video data, as well as (iii) the Google Maps API<sup>7</sup> to display geo locations in the Google Maps user interface. At least one of these APIs had to be chosen and at least two data sources had to be processed and integrated in total. As a consequence, it was allowed to add other web APIs arbitrarily. Due to our tool – respectively our prototypical implementation – being a pure data mashup tool that cannot yet handle video data and does not focus on the user interface, we used the New York Times web API and the Twitter API<sup>8</sup> as second data source for the challenge.

Based on these data sources, our scenario finds out the sentiment of people on articles of the New York Times website. To retrieve these sentiment information, we use corresponding Tweets that address the article’s topic. To achieve this, simple integration techniques are not sufficient. It is necessary to use sophisticated data analytics techniques, which can be achieved by using the concept of the previously introduced DPDs. Firstly, a named entity recognition DPD is required to search the articles for keywords that can be used to find corresponding Tweets. Furthermore, a sentiment analysis has to be conducted based on the found Tweets to receive the overall sentiment of an article. To model such a complicated data mashup scenario, the user can make use of the introduced Mashup Plans and create a graphical description using DSDs and DPDs. This description can then be used to create different executable representations as described in Sect. 4. The graphical model for this scenario is displayed in Fig. 5(a). This scenario has also been used for the runtime measurements described in Sect. 10. The execution semantics of the demo flow is described in the following Sect. 8.

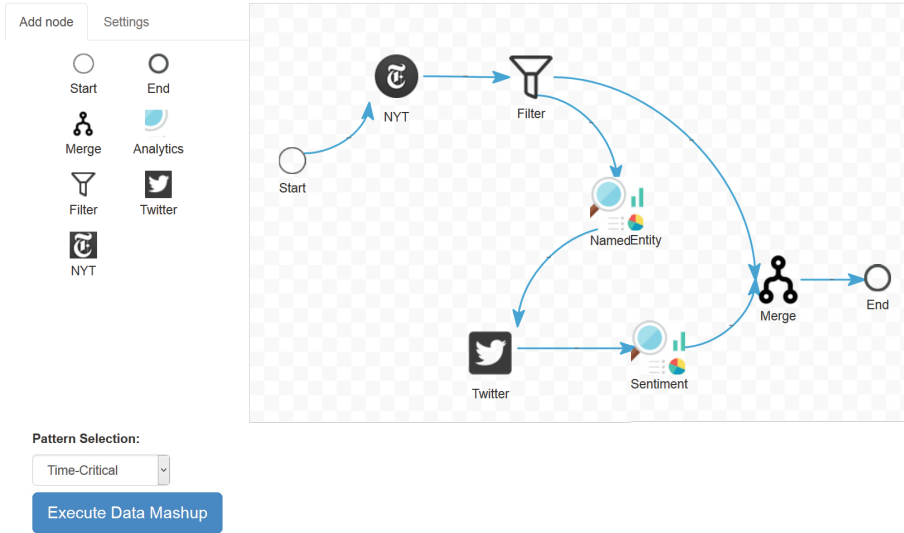
<sup>5</sup> <http://www.nytimes.com/services/xml/rss/index.html>.

<sup>6</sup> <http://www.youtube.com/yt/dev/api-resources.html>.




<sup>7</sup> <https://developers.google.com/maps/>.

<sup>8</sup> <https://dev.twitter.com/>.

# FlexMash Builder



(a) Screenshot of the demo flow used for the challenge

Article	Keywords	Overall Sentiment	Example Tweets
<p>Blue Jays 4, Yankees 0: Russell Martin Crushes Pitch, and Yankees Hopes (<a href="#">link</a>)</p>	<p>Baseball, Toronto Blue Jays, New York Yankees</p>	<p>Sentiment is Neutral!</p>	<div>  <span>WSJ Sports @WSJSports</span> <span>Follow</span> </div> <p>On an alternate universe, the Yankees might have won on wsl.com/1PwU77u</p> <p>06:18 · 14 Sep 2015</p> <p>via <a href="#">Web Street Journal</a></p>  <p>Yankees' Dream Dies After 4-0 Loss to Blue Jays</p> <p>The dream that the Yankees nursed so well over the past week—that they could come into Toronto, take two of three games, and make a legitimate challenge for the American League East title over the...</p> <p>via <a href="#">Web anshah</a></p> <p>👍 26 🌟 21</p>
			<div>  <span>SportsCenter @SportsCenter</span> <span>Follow</span> </div> <p>See ya! Russell Martin hits a 3-run homer in the bottom of the 7th and the Rogers Centre erupts!</p> <p>Blue Jays lead Yankees 4-0.</p> <p>03:41 · 24 Sep 2015</p> <p>👍 342 🌟 796</p> <p>(Sentiment Score: 0)</p>

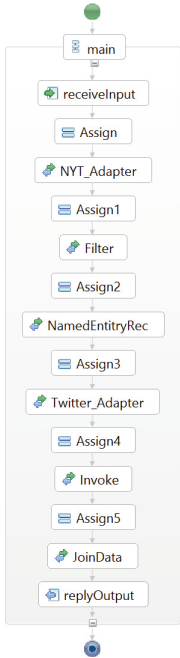
(b) Visualized result of the challenge mashup

**Fig. 5.** Screenshots of FlexMash’s Mashup Plan modeling and result view

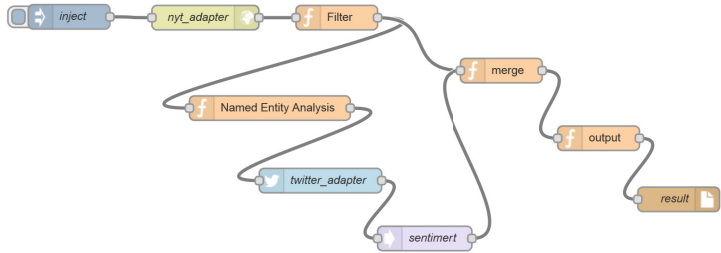
## 8 Demo Flow

Figure 5(a) depicts the flow we modeled during the ICWE Rapid Mashup Challenge. This flow is representing the scenario that was described in Sect. 7. First, a start node is added to the model. This is necessary to define the entry point of the flow-based model during execution. This start node is connected to a data

source description *NYT* that receives all current articles of a corresponding category (e.g., sports, politics, etc.) from the New York Times web API. Note that all the received articles are processed within a single flow execution. The category of the articles to be received can be configured in the node settings. Next, we connect the node to a filter that selects articles that contain certain keywords, which can also be defined in the node settings of the filter node. The filtered set of articles is sent to the merge node as well as to the *NamedEntity* node that executes a named entity recognition for each article based on the article's content to gain knowledge about its main aspects. For each article, these gained entities are then used as input for the *Twitter* search node, which is returning relating Tweets for the articles. The Twitter credentials, i.e., user name and password as well as the amount of Tweets being used have to be configured in the node settings. To do so, the credentials of an arbitrary Twitter account can be used. This configuration has to be done because Twitter demands a valid user account to access the Twitter API. For each article, the corresponding Tweets are then used for a sentiment analysis, which computes the average sentiment of all corresponding Tweets. Finally, the sentiments of the articles are merged with the article information using the merge DPD. After modeling and configuration



(a) Executable Mashup Plan as BPEL workflow



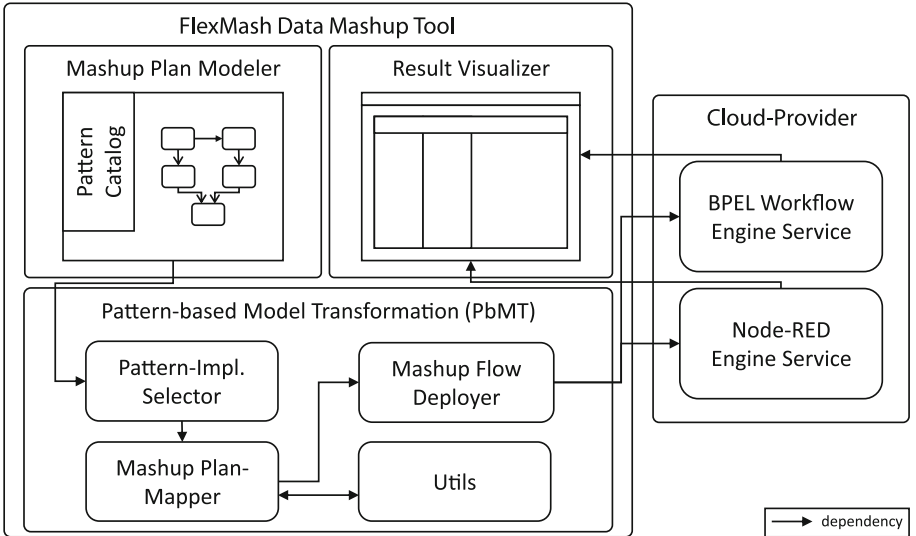
(b) Executable Mashup Plan as Node-RED flow

**Fig. 6.** Different implementations of the challenge's executable Mashup Plan

of the nodes, the user selects *Execute Data Mashup* to run the flow. The output of the flow is the generated HTML web site depicted in Fig. 5(b) containing a list of all processed articles, including their topics, the result of the named entity analysis, the computed average sentiment of an article and some example Tweets that were used for sentiment analysis. As described in Sect. 4, the Mashup Plan is transformed into executable representations based on patterns. In the demo, we implemented the robust pattern using BPEL workflows executed in the ApacheODE<sup>9</sup> workflow engine as well as the time-critical pattern using JSON-based Node-RED flows executed in the Node-RED runtime. The transformed models are depicted in Fig. 6.

## 9 Challenge Preparations

This section describes the preparations for the challenge and gives an insight into implementation details. An overview of FlexMash’s architecture specific to the implementation for the challenge is depicted in Fig. 7. Similar to other mashup tools, FlexMash is hosted online and can be accessed through a web browser. By providing FlexMash as a service on a cloud computing infrastructure (IBM Bluemix<sup>10</sup>), it can be accessed, deployed and scaled easily. The architecture contains four main components. First, the Mashup Plan Modeler (also depicted in Fig. 5(a)) that enables the user to define how the data is step-wise processed. Furthermore, in this component patterns can be viewed and



**Fig. 7.** FlexMash architecture specific to the implemented scenario

<sup>9</sup> <http://ode.apache.org/>.

<sup>10</sup> [www.bluemix.net](http://www.bluemix.net).

selected. Second, the *Pattern-based Model Transformation component (PbMT)* contains the pattern-implementation selector that automatically chooses a suitable implementation for parameterized patterns using a pattern graph-based and rule-based approach as described in Sect. 4.2. For the challenge implementation, the pattern-implementation mapping is kept simple due to the fact that only a single implementation exists for each pattern. The PbMT furthermore contains the logic of the mapping of the Mashup Plan to the executable representation depending on the pattern implementation as well as the logic of the deployment of this model onto a suitable engine. The *Utils* component contains methods supporting this functionality. The execution engines to execute the resulting model are not part of FlexMash, but cloud-based external services. Finally, the fourth component is used for visualization of the engine’s output, i.e., the execution result.

To prepare for this challenge, many implementation tasks had to be dealt with. Firstly, the existing user interface of our tool – which is based on the JavaScript framework AlloyUI<sup>11</sup> – had to be adjusted to the newly added data source descriptions and data processing descriptions used for the challenge. This task could be completed in a short time due to the framework’s high extensibility.

As described in Sect. 4, the non-executable Mashup Plan is transformed into different execution models depending on patterns that are selected after modeling. For this challenge, we implemented two mappings onto different executable models. Each of these mappings fulfills the requirements of a single pattern. The two patterns we implemented were the “*Robust Mashup*” and the “*Time-critical Mashup*” pattern. The mapping for the *Robust* pattern creates a workflow using the Business Process Execution Language as well as Java Web Services to execute DSDs and DPDs. The second mapping for the *Time-critical* pattern creates a JSON-based execution model that can be processed by the data flow engine Node-RED, which offers a very efficient flow execution. In the following, these mappings are described in a generic manner covering both execution models due to many similarities regarding the technologies being used.

To realize the mappings, first, the PbMT for the DSDs and DPDs had to be implemented. The tool’s business logic is implemented in Java and JavaScript, respectively. The data structure we use throughout the tool is JSON. The DSDs and DPDs are implemented as Java Web Services for the “*Robust*” pattern and as Node-RED JavaScript nodes for the “*Time-critical*” pattern. We implemented the NYT DSD through a HTTP request to the New York Times API to receive the RSS<sup>12</sup> feed as XML string. This string is parsed to a DOM<sup>13</sup> tree that is traversed to extract information such as the article’s title, URL, category, etc. These extracted information are stored in a JSON model for each article that is received. The Twitter DSD is implemented using the Twitter API. The DSD’s inputs are a number of keywords that are used to search corresponding Tweets. However, during the implementation, we found out that it makes sense

<sup>11</sup> <http://alloyui.com/>.

<sup>12</sup> Really Simple Syndication.

<sup>13</sup> Document Object Model.

to limit the keywords to a maximum of five. Otherwise, the search for suitable Tweets takes too much time. The received Tweets are also stored in a JSON data structure. After implementing the DSDs, we implemented the DPDs that are able to filter, merge and analyze the data.

The filter DPD used to extract articles containing specific keywords was implemented in a straight-forward manner. The article’s text as well as the title is checked for containment of the given keywords using the means of the respective programming language. If it contains one or more of the keywords, it is added to the list of filter results. Next, we implemented the two analytics DPDs, i.e., the named entity recognition and the sentiment analysis. For the named entity recognition, we used libraries provided by the Apache UIMA framework<sup>14</sup>. After a tokenization of the text, the named entities can be extracted automatically. The output of the named entity recognition is also stored in the JSON model. The sentiment analysis is conducted using the library LingPipe<sup>15</sup>. Finally, the merge node was implemented by traversing and integrating the JSON model. For the execution, we currently use execution engine services by the platform-as-a-service provider IBM Bluemix.

The result of the Mashup is visualized in a web user interface, which is based on HTML and JavaScript.

## 10 Discussion and Findings

During the implementation of our tool for the ICWE Rapid Mashup Challenge, we encountered several issues that are described and discussed in this section. We had some complications with the web APIs we used for this challenge. Firstly, we encountered the issue that the Twitter API is limited to a fixed amount of 180 Tweets every 15 min. As a consequence, we had to severely reduce the number of Tweets to be analyzed per article. However, this led to non-optimal results because usually a large amount of Tweets is necessary to compute the sentiment reliably. Furthermore, we identified weaknesses and limitations of the Twitter search and the sentiment analysis. The Twitter search seems to return advertisement, sometimes not even related to the topic we searched for. The sentiment analysis only uses single words to compute a Tweet’s sentiment without involving its context. As a consequence, the results of the Mashup we presented varied in their quality. We got some good results, however, some of the results were obviously wrong or imprecise. Due to the fact that the cause of these issues can be found in external components, the overall mashup approach could convince in regard to functionality, flexibility and powerfulness.

Table 1 displays the runtime measurements we conducted on the demo implementation presented at the ICWE Rapid Mashup Challenge. As depicted, the deployment and transformation time (which also contains the pattern-implementation selection) is nearly the same for the two execution formats being used. This can be explained by the fact that the transformation logic is very

<sup>14</sup> <https://uima.apache.org/>.

<sup>15</sup> <http://alias-i.com/lingpipe/>.

**Table 1.** Runtime Measurements

Transf. Pattern	Transformation Time Ø	Deployment Time Ø	Execution Time Ø
Robust	283,4 ms	193,8 ms	2382 ms
Time-Critical	222,8 ms	140,6 ms	23,4 ms

similar in both cases. The deployment of the executable model in the engine is also similar for these pattern implementations, however, it strongly depends on the location and efficiency of the engine being used. The main aspect, the execution time, differs greatly when comparing the two implementations. The robust execution has a high runtime due to the heavy-weight workflow engine that is being used. Additional features such as orchestration, web service calls and exception handling lead to a significant overhead. In contrast, the execution of the time-critical Mashup enables a very low runtime. This can be explained by the light-weight, JavaScript and NodeJS<sup>16</sup>-based implementation, executed in the Node-RED runtime engine, which enables efficient processing of data flows.

We are aware that the benefits of our pattern-based transformation approach do not out-stand in this use case, because there were no requirements such as robustness or efficiency defined in this challenge. However, even though the challenge use case was not completely suitable for our approach, FlexMash could convince us as well as the jurors of the challenge.

## 11 Summary and Outlook

In this paper, we presented FlexMash, an approach and tool implementation for flexible data mashups based on pattern-based model transformation. By subdividing the data mashup into four abstraction levels, namely, the modeling, transformation, execution and presentation level, we enabled an abstraction from the non-technical, domain-specific modeling of data integration and processing scenarios to the technical execution and finally the visualization and storage of the derived result. By doing so, we enabled a flexible approach through the use of transformation patterns, which leads to a data mashup execution specific to user requirements. As a consequence, we were able to create a generically applicable data mashup solution, suitable for different data sources and data processing operations usable in various use case scenarios. The evaluation of our approach was done by a prototypical implementation that was presented during the ICWE Rapid Mashup Challenge 2015 and by corresponding runtime measurements.

In the future, we are extending our pattern catalog as well as the corresponding pattern implementations. Furthermore, we will introduce *modeling patterns* to make the modeling of Mashup Plans even more domain-specific and easy-to-use.

**Acknowledgment.** This work is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) within the project SitOPT (Grant 610872).

<sup>16</sup> <https://nodejs.org/>.

## References

1. Aghaee, S., Nowak, M., Pautasso, C.: Reusable decision space for mashup tool design. In: 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2012), Copenhagen, Denmark, pp. 211–220, June 2012
2. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: portable automated deployment and management of cloud applications. In: *Advanced Web Services*, pp. 527–549. Springer, New York, January 2014
3. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA – a runtime for TOSCA-based cloud applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) *ICSOC 2013. LNCS*, vol. 8274, pp. 692–695. Springer, Heidelberg (2013)
4. Breitenbücher, et al.: Combining declarative and imperative cloud application provisioning based on TOSCA. In: *IC2E*, pp. 87–96. IEEE, March 2014
5. Breitenbücher, U., Binz, T., Leymann, F.: A method to automate cloud application management patterns. In: *Proceedings of the Eighth International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2014)*, pp. 140–145. Xpert Publishing Services, August 2014
6. Cohn, D., et al.: Business artifacts: a data-centric approach to modeling business operations and processes. *Bull. IEEE Comput. Soc. Techn. Committee Data Eng.* **32**(3), 3–9 (2009)
7. Daniel, F., Matera, M.: *Mashups - Concepts, Models and Architectures. Data-Centric Systems and Applications*. Springer, Heidelberg (2014)
8. Falkenthal, M., et al.: From pattern languages to solution implementations. In: *Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014)*, Venice, Italy (2014)
9. Hirmer, P., Wieland, M., Schwarz, H., Mitschang, B., Breitenbücher, U., Leymann, F.: SitRS - a situation recognition service based on modeling and executing situation templates. In: *Proceedings of the 9th Symposium and Summer School on Service-Oriented Computing (SUMMERSOC 2015)* (2015)
10. Hirmer, P., Breitenbücher, U., Binz, T., Leymann, F.: Automatic topology completion of TOSCA-based cloud applications. In: *Proceedings des CloudCycle14 Workshops auf der 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*. LNI, vol. 232, pp. 247–258. Gesellschaft für Informatik e.V. (GI) (2014)
11. Hirmer, P., Reimann, P., Wieland, M., Mitschang, B.: Extended techniques for flexible modeling and execution of data mashups. In: *Proceedings of the 4th International Conference on Data Management Technologies and Applications (DATA)*, April 2015
12. Kassner, L.B., Mitschang, B.: MaXCept-decision support in exception handling through unstructured data integration in the production context. An integral part of the smart factory. In: *Proceedings of the 48th Hawaii International Conference on System Sciences* (2015)
13. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – a modeling tool for TOSCA-based cloud applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) *ICSOC 2013. LNCS*, vol. 8274, pp. 700–704. Springer, Heidelberg (2013)
14. Künzle, V., et al.: PHILharmonicFlows: towards a framework for object-aware process management. *J. Softw. Mainten. Evol.: Res. Pract.* **23**(4), 205–244 (2011)
15. Meunier, R.: The pipes and filters architecture. In: *Pattern Languages of Program Design*, pp. 427–440. ACM Press/Addison-Wesley Publishing Co. (1995)
16. OASIS: *Topology and Orchestration Specification for Cloud Applications* (2013)

17. OASIS: TOSCA Primer, November 2013. <http://docs.oasis-open.org/tosca/tosca-primer/v1.0/cnd01/tosca-primer-v1.0-cnd01.pdf>
18. Reimann, P., et al.: Data Patterns to Alleviate the Design of Scientific Work Flows Exemplified by a Bone Simulation. In: Proceedings of the 26th International Conference on Scientific and Statistical Database Management (2014)
19. Reimann, P., Schwarz, H., Mitschang, B.: A pattern approach to conquer the data complexity in simulation workflow design. In: Meersman, R., Panetto, H., Dillon, T., Missikoff, M., Liu, L., Pastor, O., Cuzzocrea, A., Sellis, T. (eds.) OTM 2014. LNCS, vol. 8841, pp. 21–38. Springer, Heidelberg (2014)

Rapid Mashup Development Tools

First International Rapid Mashup Challenge, RMC 2015,

Rotterdam, The Netherlands, June 23, 2015, Revised

Selected Papers

Daniel, F.; Pautasso, C. (Eds.)

2016, VII, 123 p. 52 illus., Softcover

ISBN: 978-3-319-28726-3