Semantic Mediation Techniques for Composite Web Applications

Carsten Radeck, Gregor Blichmann, Oliver Mroß, and Klaus Meißner

Technische Universität Dresden, Germany
{carsten.radeck,gregor.blichmann,oliver.mross,
klaus.meissner}@tu-dresden.de

Abstract. The mashup paradigm allows end users to build custom web applications by combining data-exchanging components in order to fulfill specific needs. Since such building blocks typically originate from different third party vendors, compatibility issues at component interface level are inevitable. This decreases re-usability and requires skilled users or automatisms to provide the necessary mediation to solve such issues. However, current mashup proposals are very limited in this regard.

We present techniques for data mediation that leverage semantically annotated interface descriptions to overcome a high degree of interface mismatch. We equipped the EDYRA mashup platform for end user development with automatic support for these techniques to increase the re-usability of components and to foster the long tail of user needs. In order to show the practicability of our approach, we describe the platform implementation and present benchmark results.

Keywords: mashup, semantics, data mediation, end user development.

1 Introduction

Recently, universal composition approaches like CRUISe [1] allow for platform-independent modeling of mashups and a uniform description of components spanning all application layers. Since components are typically developed by different third party providers, combining component interfaces in a meaningful way is far from trivial. Data exchanged between components may differ in various aspects leading to incompatibilities: providers use different vocabularies, schemata, units or abstraction levels when designing interface signatures. This complicates end user development (EUD) further, and connecting components in ways not anticipated becomes a cumbersome task. Semantic technologies are a potent solution to provide **data mediation**, i. e., automatic resolving of heterogeneous data structures. Although proposals in the semantic web service (SWS) domain exist, most mashup platforms neglect data mediation so far.

Within the EDYRA project, we adhere to universal composition and strive for enabling domain experts without programming skills to build and reuse composite web application (CWA). We utilize semantic annotations to refer to ontology concepts of component interfaces. Based on this, semantic data mediation techniques are applied by our platform and hidden from end users.

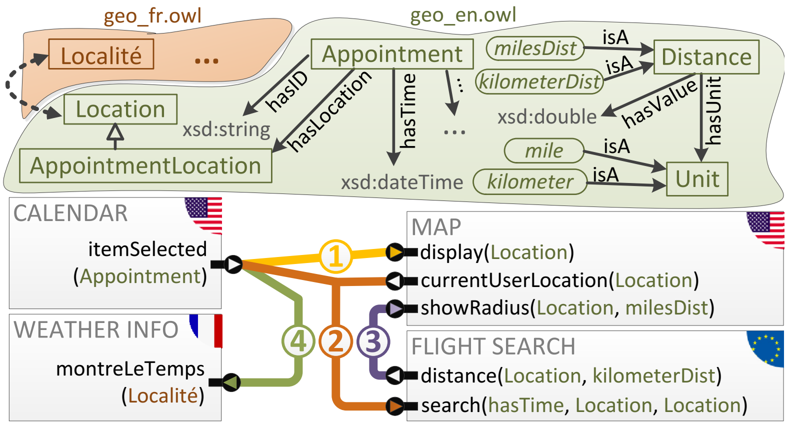

Fig. 1. Reference scenario: conference planning

To highlight arising challenges, let us consider the following use case which we use as a **reference scenario** throughout this paper.

Non-programmer Bob from the USA builds a CWA (Fig. 1) to organize a conference participation in Toulouse, France. He selects the conference, which he noted previously as an appointment, in the calendar. Then Bob wants to see the appointment location on a map. However, syntactically there is no possibility to combine the calendar and the map component's interface. While the calendar offers a data object of type `Appointment`, the map consumes a `Location`. Therefore, semantically it would be possible to take the appointment's `AppointmentLocation` and "cast" it to a general `Location` ①. Next, he wants to search for flights to Toulouse ②. In this case, he adds a flight search service and uses his current location as well as the appointment's location and time as search criteria. Besides the semantic problem of querying time and location of the appointment, it is necessary to put all three parameters together in compliance with the signature required by the flight service. Furthermore, Bob wants to visualize the distance between the airport and the conference location, which is calculated by the flight service ③. Because the flight service is located in Europe, the distance is provided in kilometers. To use the `showRadius` functionality of the map (from an American provider) this value has to be converted to miles. Finally, Bob wants to see the weather forecast for the appointment's time and location. He utilizes a French weather service which uses an ontology for annotating the interface that differs from the calendar's ④. But semantically the same concepts are described and the components can be coupled.

Current mashup proposals lack capabilities to implement this scenario. Thus, as our main contribution, we introduce data mediation techniques for CWAs and show their practicability within our EUD platform. These concepts help to combine components in more flexible ways than pure syntactic interfaces would allow, increasing re-usability and fostering the long tail of user needs.

The remaining paper is structured as follows. Sect. 2 presents mediation techniques for CWA. In Sect. 3, we describe our mashup platform with mediation support and show the practicability of our concepts. In Sect. 4, we discuss related work. Sect. 5 concludes the paper and outlines future work.

2 Semantic Data Mediation Techniques for CWA

Data mediation serves to resolve interface incompatibilities, of course within certain boundaries. In this section, incorporating results from the SWS domain like WSMO Mediators [6], we introduce a set of generic mediation techniques for CWA. We apply semantic data mediation and thereby leverage the domain knowledge defined in OWL-DL ontologies and annotated to component interfaces [1]. Essentially, annotations refer to classes, datatype and object properties or individuals in ontologies modeling the application/component domain. Since it is possible to model the same domain in various ways, we assume a certain modeling and annotation style.

In general, ontology classes can be annotated directly, e. g. `Location`, or via an OWL object property whose range the class is, if it is necessary to highlight a more specific meaning, e. g. `hasCenter`. Additionally, OWL datatype properties can be used, e. g. `hasLatitude`. However, there are circumstances where it is more appropriate to model individuals rather than subclasses. Units, currencies, and quantities, i. e., convertible concepts, may be mentioned as examples, or classes that refer to such convertibles on OWL property level (see `Distance` in Fig. 1). In this case, concrete individuals should be annotated, e. g. `milesDist`, a `Distance` individual where `hasUnit` points to `mile`.

In general, data transfer is realized through interface elements, which are *properties*, *operations* and *events* in our case. Interface elements can have one (e. g. properties) or more (e. g. operations and events) parameters, which have an identifier and a semantic type annotation. Channels combine one interface element of a source component SC with one of a target component TC. Therefore, an assignment $assi$ has to exist, that maps n parameters P_{out} of the SC bijectively to all n parameters P_{in} of the TC. A perfect match exists if $assi$ only includes mappings between parameters that are semantically identical (both refer to the identical concept). As an example, the mapping `(latitude, longitude)` $\rightarrow$ `(latitude, longitude)` is a perfect match.

Due to the usage of third-party components, a perfect match is unlikely. P_{out} and P_{in} can be *semantically compatible* if a **Semantic Connector** $SeCo$ can be defined, which is a set of channels and mediation techniques. It ensures that all parameters P_{in} of *one* interface element of a TC are connected and of the required semantic type. This may include that several channels from one or more SC can exist. In case of multiple inbound channels, the $SeCo$ takes care of an appropriate synchronization between them.

Upcast. As proposed earlier [1], the *upcast* mediation technique serves for solving different generalization levels of concepts annotated at parameters. In case of classes this means, that a more specific class is cast into a more generic one as long as they are in `subClassOf` relationship. Assume that a component outputs an `AppointmentLocation` but the target component requires a more generic `Location`. Then a upcast can be applied.

In principle, upcasts may additionally be used for OWL object properties if there is a `subPropertyOf` relation, by dealing with it as if the range would have been annotated. In case of datatype properties we presume that the underlying range stays the same, rendering upcasts simple. Upcasts are one way, i. e., only casts upwards the inheritance hierarchy are valid along the data flow.

Conversion. The *conversion* mediation technique has two main application areas. First, it resolves incompatibilities between two parameters annotated by convertible concepts, like *units, quantities* and *data types*. Please consider the example `kilometer` → `mile` from Fig. 1. The specific knowledge required can, e. g., be formalized in dedicated ontologies like the QUDT (`http://qudt.org`). In the latter, a base type is assigned to each unit for conversion purposes. For example, `meter` is the base type for `LengthUnit`, to which all other length units have a conversion factor to. Another use case are *scale adjustments* requiring more domain-specific transformations, e. g., mapping a five star rating to a ten point rating. Typically, these conversions require domain-specific knowledge and cannot be covered by generic algorithms or reasoners.

Second, conversion is used on class level in case of `equivalentClass` relationships, e. g. `Location` and `Localité` in Fig. 1 or `Location` and `Place`. For sake of simplicity, we pose a rather strict definition of equivalence: There have to exist `equivialentProperty` relations for all declared properties of those classes.

Semantic Split. A *semantic split* queries multiple OWL properties of an individual, which is represented as a parameter or property, and distributes them on one or more parameters of a target interface element. Fundamentally, only individuals can be "split" within the restrictions of their ontology class. This mediation technique is applicable if the following OWL constructs are annotated:

- Class: OWL data and object properties can be assigned to target parameters that reference a semantically compatible class, data or object property, e. g. connection ① in Fig. 1 (`Appointment.hasLocation → Location`).
- OWL object property: This case is handled as if the range class of the object property is annotated, e. g. `hasLocation` → {`hasLatitude, hasLongitude`}

Semantic Join. A *semantic join* creates an individual, representing a target parameter, by joining of multiple parameters of one source interface element. Assume, that a map publishes an event with parameters {`hasLatitude, hasLongitude`} and there is a point of interest finder that offers an operation consuming a parameter of type `Location`. Then, a semantic join is possible.

It has to be guaranteed, that the generated individual fulfills all constraints on OWL properties defined by the target class (and thus all superclasses).

Partial Substitution. Using a *partial substitution*, an OWL property of an individual represented as parameter can be updated with an individual or literal given by another parameter. With regard to Fig. 1, a partial substitution is possible between `Location` and `Event`, since the object of the OWL property `hasLocation` of an `Event` can be substituted by a `Location` individual. Partial substitutions are exclusively applicable for properties as target interface element. This is caused by the fact, that in our component model only properties expose and allow to change a partition of the component's data layer directly.

Using partial substitution increases the possibility to connect properties bidirectionally. As an example, consider that the calender has a property for its currently selected appointment and the map has a property for the currently selected location. Beside the possibility to semantically split the event to display its location, it is even feasible to connect the map to the calendar, to substitute the appointment's location with that of the map by dragging the map's marker.

Partial substitutions are not suitable for connections between events and operations for two reasons: First, it is not guaranteed that an event is correlated to an input interface element which can update the individual represented by the event. Second, events and operations in general hide the data layer.

Syntactic Join. A *syntactic join* is intended to synchronize `m` parameters published by `n` source interface elements of several *SC* and feeds them together in `1` target interface element. Several synchronization modes are supported.

- *tolerant*: The joiner waits until all sources have published at least once. Only the latest parameters are cached per source (the old value is overridden), and the cache is cleared after data transfer to the target.
- *repeating*: Here, the cache is not cleared, i. e., once all sources have sent data, each following publication causes the joiner to transfer data to the target.
- *queuing*: There is a queue per parameter. When all queues have at least one entry, the data is transferred to the target and the first element is removed.

3 Mediation-Equipped Platform for Mashups

3.1 Architecture

Our platform builds up on the CRUISe and EDYRA infrastructure we introduced earlier [1,2]. An overview is shown in Fig. 2. Universal composition is applied to create and execute presentation-oriented CWA, where components of the data, business logic and user interface (UI) layer share a generic component model. The latter characterizes components by means of several abstractions: parametrized events and operations, properties, and capabilities. The Semantic Mashup Component Description Language (SMCDL) serves as a declarative language implementing the component model. It features semantic annotations to clarify the meaning of component interfaces and capabilities [2]. Based on the component model, the declarative Mashup Composition Model (MCM) describes all aspects of a CWA, e. g. included components and event-based communication.

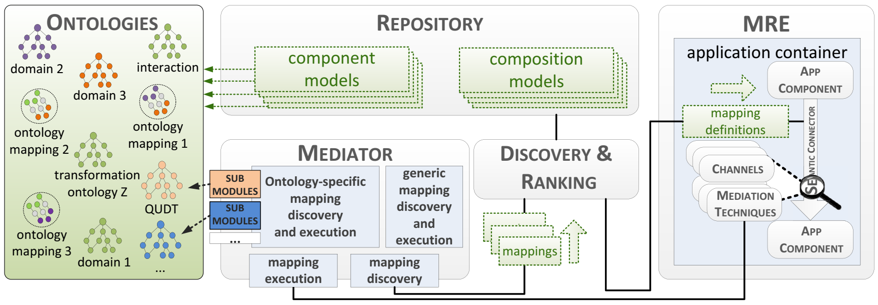

Fig. 2. Architectural overview of our mediation-equipped mashup platform

A *repository* is in charge of managing components and compositions. Furthermore it provides services for querying those artifacts.

Ontologies play an important role in our approach as they serve for annotating components and provide the schema knowledge most mediation techniques are based on. Besides domain-specific ontologies there can be (1) upper ontologies, e. g., for units, (2) ontologies defining how to transform concepts, and (3) mapping ontologies pointing out similarity relations of concepts in different knowledge representations. For the provision of mapping ontologies we assume that a state-of-the-art ontology alignment process takes place. Only confirmed mappings result in an ontology, linking concepts via predicates like `equivalentClass` and `sameAs`, e. g. `Location equivalentClass Localité`. A component can introduce new ontologies by using them for semantic annotation.

To enable recommendations, there are facilities for *discovery and ranking* of composition fragments, which represent composition knowledge, best matching user requirements and the current context. Within those modules, algorithms for recommendations utilize amongst others the semantic component annotations and ontology mappings. An essential task during discovery is the calculation of semantic connectors between components. Thereby, *mapping definitions* are derived utilizing the mediator or reused if already calculated.

A *mapping definition* specifies the mediation techniques required to align interface elements in a semantic connector. It is a data structure consisting of: an *ID*, P_{in} and P_{out} for more efficient matching and reuse, and one or more *mappings* including the composition and configuration mediation techniques, like the synchronization mode of a syntactic join.

A *mashup runtime environment (MRE)* interprets composition models in order to run mashups. With regard to data mediation, an MRE provides automatic support for the proposed mediation techniques, see Sect. 3.2 for details.

The *mediator* is responsible for mainly two tasks. First, it provides means for looking up mappings, which involve mediation techniques. To this end, the mediator takes two signatures, i. e., the URIs of concepts annotated at a source and a target interface element. In order to detect mappings e. g. between `milesDist` and `kilometerDist` in our scenario, algorithms have to inspect concepts on OWL property level, whereby we restrict the depth to one level. This task may result in multiple valid mappings, which have to be ranked further. Second, the mediator serves for the execution of mediation techniques defined in mapping definitions as requested by an MRE.

As illustrated in Fig. 2, there are generic and ontology-specific algorithms to accomplish these tasks. Generic algorithms utilize standard modeling constructs and reasoning rules of RDF/S and OWL. Mainly, relations like subsumption, property ranges and domains, `sameAs` and `equivalentClass` are leveraged.

The mediator can be extended with ontology-specific modules by implementing the required interface. They use dedicated modeling constructs and reasoning rules to derive and apply mappings and are consulted if generic algorithms cannot provide a mapping. Knowledge is encapsulated in modules for the algorithmic part and the corresponding ontologies for the terminology. Although in principle generically applicable, such modules are especially useful for conversions. There are modules per transformation ontology, responsible for interpreting and applying transformations on the payload delivered by events/properties. To identify suitable modules, each one provides a method to state if it supports the given signatures during discovery as well as execution of mappings.

Within the EDYRA project[1], we implemented a client-side thin-server MRE completely written in JavaScript. The mediator is distributed over the MRE and the SOAP-based *mediation service*. The latter is implemented in Java and uses the Jena framework for working with semantic models, including validating, reasoning and querying via SPARQL. The mapping discovery is located at the Java-based repository. We implemented the `DataSemanticsMatcher` as a generic algorithm that builds up on a QUDT-specific and a generic conversion module. The QUDT module supports annotated parameter types which refer to QUDT concepts. It queries the QUDT ontologies to check if both concepts are convertible, i. e., if they belong to the same unit domain. At execution time, the conversion multiplier is looked up and applied. The generic conversion module utilizes OWL constructs like `equivalentClass` and `equivalentProperty` to decide if two given classes are equal according to our definition in Sect. 2. To add new ontologies in our prototype, they have to be used for semantic annotation in SMCDL and registered manually at the repository and mediation service.

3.2 Runtime Support

Semantic connectors are implemented by associating the mapping definitions of semantic mediation techniques with a communication channel, and providing syntactic joins as built-in mediation components. The latter comply to the component model and can consequently be connected with other components. The MRE has templates for the SMCDL and the implementation of joiners and configures those as stated in the mapping definition to seamlessly instantiate and manage joiners like application components. With syntactic joins as components, single channels connect one SC with one TC.

[1] `http://mmt.inf.tu-dresden.de/edyra` (also links to our live demonstrator).

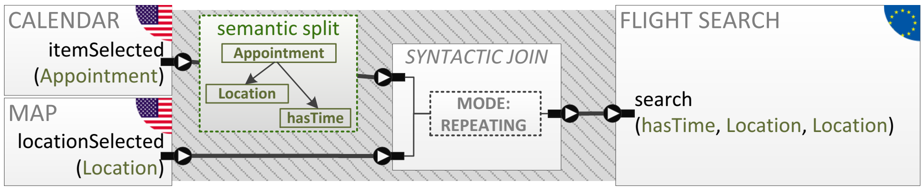

Fig. 3. Implementation of an exemplified semantic connector from the scenario

Fig. 3 shows an example (dashed area) of a semantic connector present in the mashup from our scenario. The semantic split is realized through a mapping definition which is directly attached to the channel between the calendar and a syntactic join. The latter aligns the split `Location` and `Date` as well as the `Location` from the map and transfers them combined to the flight search.

As described in our previous work [1], our components predominantly exchange data serialized in XML. There is a predefined *grounding* per ontology concept specifying the XML schema for individuals and literals. In case component developers utilize their own schemata or even other formats, like JSON, they have to ensure that data are transformed to the grounding, either as part of the component implementation or by transformation instructions in the SMCDL. In order to apply semantic mediation, data has to be available in RDF. Thus, we assume lifting and lowering transformations per predefined grounding.

When a *SC* publishes data according to the grounding, the channel delegates the execution of mediation techniques to the mediator by handing over the transported payload, the mapping definition, and in case of partial substitutions the target property's value. Per parameter, the mediator applies the lifting to get triples, which are then put in separate semantic models together with the terminological knowledge of ontologies. Using Jena in our prototype, validation and reasoning takes place automatically. Then the mediation techniques are executed as configured by the mapping definition. We utilize Jena's OWL API, e. g., to add OWL property values to individuals, and invoke conversion modules if required. Next, lowering takes place. Thereby, per target parameter, the model is queried with SPARQL, the results are serialized in the XML results format on which an XSLT transformation is applied. Finally, the mediated payload is forwarded to the *TC*. For simple datatypes as grounding, e. g. as input for conversions or result of a split, there is no dedicated lifting and lowering required. We programmatically map primitive data types to `Literals` and vice versa.

Syntactic joiners are connected to `n` inbound and one outbound channels. To achieve that, there are `n` operations in the joiner's generated SMCDL, whose parameter count and names correspond to those of the source interface element. Occurring event on an inbound channel are handled by the connected operation. The joiner extracts the parameter payload, handles it (e. g., adds it to a queue) and decides whether to fire its event according to its synchronization mode.

We conducted a benchmark where we measured the average response time of the mediation service's operations implementing the mediation techniques.

Table 1. Average response time of the mediation service in 100 runs per technique on a local server. The setup includes an Intel Core i7 2.8 GHz and 32 GB RAM.

Mediation technique	**Avg. response time**
Upcast (`AppointmentLocation` → `Location`)	≈15 ms
Conversion (`kilometers` → `miles`)	≈14 ms
Conversion (two equivalent classes)	≈40 ms
Semantic split (`Location` → {`hasLat, hasLng`})	≈23 ms
Semantic join (reversed split)	≈13 ms
Partial substitution (`Location` → `Event.hasLocation`)	≈23 ms

The results show a decent performance considering that lifting and lowering takes place in most cases. Network overhead of SOAP over HTTP may result in noticeable delays on slow connections, so that user experience may suffer in comparison to perfect match channels. That can be lowered by using WebSockets, and by integrating the mapping execution in a client-server MRE. Other crucial factors are ontology size and complexity, especially when a reasoner is attached.

4 Related Work

Proposals for semantically annotated services mostly use lifting and lowering to transfer data to the semantic layer [4,5]. While we use this, because our components do not exchange semantic data, our concept is not limited to upcasts.

Research shows that semantic web service descriptions are suited for mediation. There are different mediators in the conceptual framework of WSMO [6], which are provided as web services too and completely operate at a semantic layer. There are similar techniques involved, but due to the lack of a UI, execution performance is not that critical as for CWA. In addition, only a `1:1` communication is supported, while we can handle `n:m` semantic connectors.

For mashups, automated data mediation has been neglected so far [7]. Simple constructs exist that support, e. g., filtering, assignment and sorting of data, for example in Yahoo Pipes. But those rather belong to application logic than generic mediation techniques, and data semantics is not taken into consideration. Few approaches use semantically annotated component interfaces for matching at all, like [8,1]. Our previous work [1] can solve syntactical issues like different parameter naming with the help of wrappers. As a semantic issue only upcasts can be handled. Therefore, we largely extend this work.

5 Conclusion and Future Work

Since components of a CWA typically originate from different vendors, connecting them in meaningful way is challenging. Incompatibilities of signatures and exchanged data may, for instance, result from varying vocabularies or units. This complicates EUD, especially for non-programmers. In addition, connecting components in unforeseen ways becomes far from trivial. Thus, platforms for mashup EUD should feature means to automatically provide the required glue code.

We describe a set of data mediation techniques that use semantic component annotations to resolve interface mismatches. Those techniques reflect knowledge captured by ontologies, can be combined and are essential for establishing semantic connectors between components. The higher flexibility for combining components fosters re-usability and unforeseen coupling. This way, more niche requirements can be meet without the need for new components. Utilizing our implemented core platform, we show the practicability. However, due to the increased possibility of combinations, it is challenging to identify useful connectors. In addition, our approach depends on semantic annotations of OWL concepts, and we limit the depth to 1 when analyzing concepts during mapping discovery. Tough this may restrict the solution space, it lowers the algorithmic complexity.

Currently, we utilize mediation techniques for deriving and visualizing recommendations, in the CapView [2] and for synchronization of mediable components during collaboration. Future research will focus on the mediation of collections.

Acknowledgments. Carsten Radeck and Gregor Blichmann are funded by the German Federal Ministy of Economic Affairs and Energy (ref. no. 01MU13001D).

References

1. Pietschmann, S., Radeck, C., Meißner, K.: Semantics-based discovery, selection and mediation for presentation-oriented mashups. In: 5th Intl. Workshop on Web APIs and Service Mashups (Mashups), pp. 1–8. ACM (September 2011)
2. Radeck, C., Blichmann, G., Meißner, K.: Capview – functionality-aware visual mashup development for non-programmers. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 140–155. Springer, Heidelberg (2013)
3. Shvaiko, P., Euzenat, J.: Ontology Matching: State of the Art and Future Challenges. IEEE Transactions on Knowledge and Data Engineering 25(1), 158–176 (2013)
4. Szomszor, M., Payne, T., Moreau, L.: Automated syntactic medation for web service integration. In: Proc. of the Intl. Conf. on Web Services, pp. 127–136 (September 2006)
5. Nagarajan, M., Verma, K., Sheth, A.P., Miller, J.A.: Ontology driven data mediation in web services. Intl. Journal of Web Services Research 4(4), 104–126 (2007)
6. Mocan, A., Cimpian, E., Stollberg, M., Scharffe, F., Scicluna, J.: WSMO mediators (December 2005), `http://www.wsmo.org/TR/d29/`
7. Di Lorenzo, G., Hacid, H., Paik, H.Y., Benatallah, B.: Data integration in mashups. SIGMOD Rec. 38(1), 59–66 (2009), `http://doi.acm.org/10.1145/1558334.1558343`
8. Bianchini, D., De Antonellis, V., Melchiori, M.: A recommendation system for semantic mashup design. In: Workshop on Database and Expert Systems Applications (DEXA), pp. 159–163 (September 2010)