Towards Sharable Application Ontologies for the Automatic Generation of UIs for Dialog based Linked Data Applications

Michael Hitz[1], Thomas Kessel[1] and Dennis Pfisterer[2]
[1]*Cooperative State University Baden-Wuerttemberg, Stuttgart, Germany*
[2]*Institute of Telematics, University of Lübeck, Lübeck, Germany*

Keywords: User Interface Ontologies, Model Driven User Interfaces, Linked Data Application Modelling.

Abstract: The emerging *Internet of Everything* is a driving force for businesses to expose their processes as services to third parties to be integrated into their applications (e.g. the booking of a trip or requesting the quote for a complex product). To standardize the processes and related data, increasingly semantic web technologies are applied - leading to a shared conceptualization of the business domains and thus creating a *linked data service ecosystem* for domain-specific services. Although the communication on machine-level is standardized by using semantic web technologies, the integration of the user into the overall process is still a manual task: User Interfaces (UI) for collecting the input data for a process are built manually for multiple platforms and user groups. The claim of this paper is, that given a linked data service ecosystem, UIs can be modelled and automatically generated for integration into linked data applications. The paper presents an ontology-based, model-driven approach for modelling UI variants for automatically generating dialog-based applications, providing output understood by associated linked data services.

1 INTRODUCTION

The trend toward the digitalisation of business processes and the need to expose business functionality via multiple channels to different user groups led to a strong adoption of service oriented concepts for enterprise information systems. Companies offer services (e.g., as web services) that are driven by user inputs to invoke processes like ordering goods or services. This opens new business opportunities for third parties that aggregate such services to novel applications. Prominent examples are *Uber* (developer.*uber.com*) or *Amazon Marketplaces* (developer.amazonservices.*com*) who expose their offerings through proprietary APIs.

Emerging business models such as *Distributed Market Spaces* in an *Internet of Everything* (IoE) context (e.g., Radonjic-Simic et al., 2016) go even further and use a generic, non-proprietary data format. They incorporate semantic web / linked data approaches (i.e., ontologies) to describe the semantics of the expected input data and thus create a shared conceptualization of the domain. This allows multiple suppliers to participate in a transaction (e.g., providing information for the comparison of offerings) relying on the same input data and based on strictly defined semantics leading to a unified view on the processes; and thus create a ***linked data services ecosystem***. The industry begins adopting these principles by defining reliable data interchange semantics for different domains (e.g., the BiPRO initiative, www.bipro.net that standardizes business processes and data for the insurance sector).

Albeit there exists **a clear concept on the technical level for machines** to work on and communicate with semantically specified data (i.e., linked data technologies such as RDF/OWL) there is still a **lack of approaches for the integration of the *human user.*** Non-trivial user interfaces (UIs) are needed to collect user input for the business processes while supporting a platform-specific user experience (e.g., web frontends, mobile apps or rich client desktop apps). To support the specific needs of user groups and different platforms in a multi-channel environment, different variants of user interfaces are required. Currently, these UI variants are mostly developed manually for each application context.

Given the above mentioned *environment of a linked data ecosystem as a prerequisite*, the basic assumption of the presented paper is that UIs for dialog based linked data applications can be **(1) automatically generated** serving linked data ser-

Hitz M., Kessel T. and Pfisterer D.
Towards Sharable Application Ontologies for the Automatic Generation of UIs for Dialog based Linked Data Applications.
DOI: 10.5220/0006137600650077
In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, pages 65-77
ISBN: 978-989-758-210-3

vices and can be **(2) reused and shared in different contexts** (e.g., portals or rich client applications).

Linked data services provide a definition of the semantics of the expected input data using ontologies. Hence, UIs that provide the required input data can be used as a frontend for these services. And if UIs are modelled in a technology-agnostic way this results in UI descriptions that can be shared and reused.

To automatically generate UIs for that purpose, (1) an abstracted UI description is needed to generate UIs for different technical platforms. To make the description sharable, it (2) needs to be modelled in a non-proprietary, standardized way. Finally, to be used in conjunction with linked data services, (3) the UI needs to produce an output that conforms to the input of these services.

This paper proposes an approach, which addresses these requirements: it uses sharable Application Ontologies containing the necessary information to derive UIs for different contexts and to produce linked data requests as outcome.

Although there exist approaches for model-driven UI generation (cf. related work, Section 8), there is - to the best of our knowledge - no widely accepted approach or UI modelling technique that solves the aforementioned issues (cf., Meixner et al., 2011). Traditional approaches (e.g., user interface description languages - UIDL) rely on proprietary UI technology focused models. They are strong in producing technological variants of UIs. The downside in their applicability in a linked data context is the proprietary, UI-focused nature of the modelled artefacts, which impedes their use in different contexts (Coutaz, 2010). In addition, the mapping of input to target data - if possible at all - encompasses the creation of many related artefacts and thus being very complex (e.g. *UWE* - Kraus et al., 2003). Research regarding the ontology-based generation of UIs mainly focuses on providing editors for editing instances of arbitrary ontologies. These generic interfaces are technical in nature and not suitable for presentation to a customer as they do not focus on user experience.

The paper presents a novel approach for the automatic UI generation of linked data applications that bridges the gap between the traditional and ontological approaches. It contributes to the field of automatic UI generation applied to linked data concepts.

The paper is structured as followed: First, the problem is demonstrated in more detail along with an illustrative example used throughout the paper. Section 3 outlines the proposed solution. Section 4 and 5 provide details for the proposed Application Ontology. Section 6 outlines the process for derivation of UIs and resulting instance data followed by the current state of evaluation of the concept. The paper closes with related work and conclusion pointing out future work.

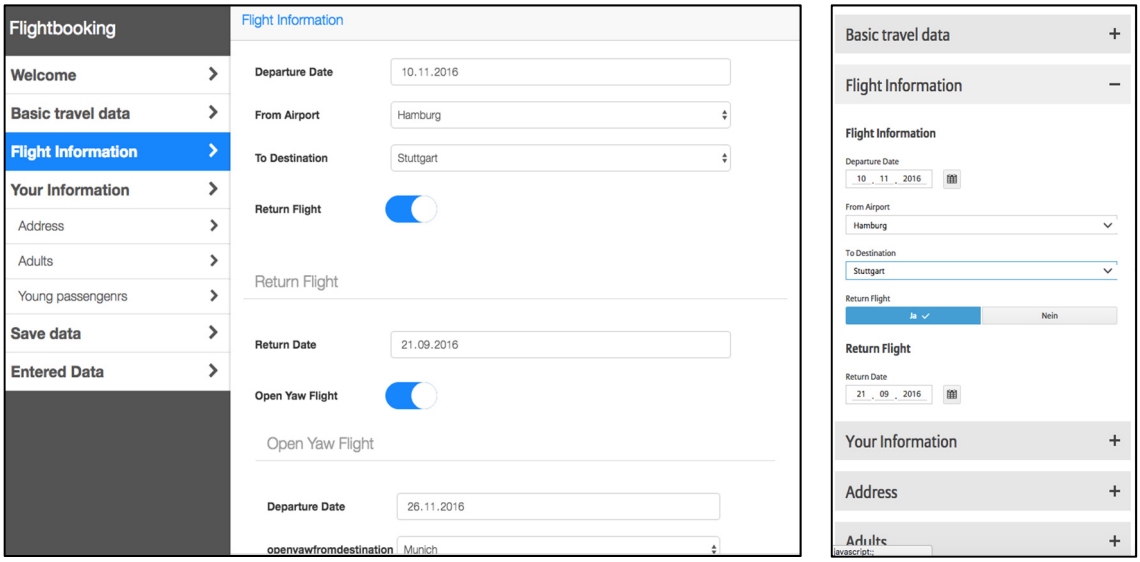
2 PROBLEM, MOTIVATIONAL EXAMPLE & REQUIREMENTS

To generate high-quality UIs for dialog applications, a pure data-model, like a model of the input data for the underlying business process does not suffice. For example, UIs usually group information in a meaningful way. The structure of questions is usually different from the target data structure of a consuming service. Most UIs include dynamic behaviour to guide the user through the data gathering process in an intuitive way (e.g., prefilling related information as a city name given a zip code or showing / hiding information based on provided data). The information needed to build these aspects are usually not part of a data-model (Hitz, 2016) and thus need to be modelled separately.

Example: Consider a (simplified) process collecting quotes for a ***flight booking***. A customer requests quotes and thus needs to specify data about the flight. He supplies information about dates, number of tickets, return/open-yaw-flight and customer-related information (e.g., name, billing address etc.)

An excerpt of the possible request data (*target data*) based on a user's input is shown in Listing 1. It is intentionally simplified and represented in RDF/Turtle notation as instance of an (assumed) flightbooking ontology. It contains information about the flight (3 tickets from Hamburg to Stutgart), return flight (to Hamburg) and an open-yaw-flight along with data about the customer.

Fig. 1 shows possible UI variants for the user input dialogs: (a) a desktop application for an agent and (b) a mobile application for end customers. The information is structured in **meaningful succession of groups and questions** (e.g., *Basic travel data*, *Flight Information* and *Your information*) and might have **hierarchical relations** (e.g., *Address* data being part of *Your information*). The questions are presented in a reasonable order, using **type-related input** controls allowing an intuitive user interaction. In addition to these *structural aspects*, the UI needs to offer *dynamic functionality* for a satisfying user experience: input needs to be **validated** and errors shown (e.g., if the *return date* is before the *departu-*



a) Desktop version (agent) b) Mobile version (cutomer)

Figure 1: Possible UIs for the flight booking application sample.

re date), data might be prefilled as **reaction to previous input** (e.g., restricting *destination airports* that are in served by an already selected *departure airport*) and information should be **shown / hidden based on previous selections** (e.g., hiding *open-yaw-* or *return flight related information* if the user deselects these options).

Listing 1: Instance data for a flight request.

```
@prefix : <http://mimesis.solutions/bookers/flightbooking/individuals#> .
@prefix owl: ... ,rdf: ..., xml:..., xsd, ..., rdfs:...
@prefix fbo: <http://mimesis.solutions/bookers/flightbooking/v1#> .
@prefix foaf: <http://xmlns.com/foaf/0.1#> .
@base <http://mimesis.solutions/bookers/flightbooking/individuals> .
<http://mimesis.solutions/bookers/flightbooking/individuals>
   rdf:type owl:Ontology .

:flightbookingrequest_i1474371413428
   rdf:type <fbo:FlightBookingRequest> , owl:NamedIndividual ;
  <fbo:childtickets> "1"^^<xmls:number> ;
  <fbo:adulttickets> "2"^^<xmls:number> ;
  <fbo:customerinfo> :customerinfo_i1474371413428 ;
  <fbo:flight> :flight_i1474371413428 .

:flight_i1474371413428 rdf:type <Flight> ,owl:NamedIndividual ;
 <fbo:returndate> "2016-09-20T22:00:00.000Z"^^<xmls:date> ;
 <fbo:startdate> "2016-11-09T23:00:00.000Z"^^<xmls:date> ;
 <fbo:fromdestination> "HAM"^^<xmls:string> ;
 <fbo:todestination> "STR"^^<xmls:string>;
 <fbo:openyawstartdate> "2016-11-25T23:00:00.000Z"^^<xmls:date> ;
 <fbo:openyawtodestination> "HAM"^^<xmls:string> ;
 <fbo:openyawfromdestination> "MUC"^^<xmls:string> .

:customerinfo_i1474371413428 rdf:type <Customerinfo> , ... ;
 <foaf:givenName> "Max"^^<xmls:string> ;
 <foaf:familyName> "Mustermann"^^<xmls:string> ;
 <foaf:gender> "male"^^<xmls:string> ;
 <foaf:email> "max.mustermann@onemail.com"^^<xmls:string> ;
 <fbo:billingaddress> :billingaddress_i1474371413428 .

:billingaddress_i1474371413428 rdf:type <fbo:BillingAddress> , ... ;
  <foaf:buildingNo> "178"^^<xmls:string> ;
  <foaf:zip> "70178"^^<xmls:string> ;
  <foaf:street> "Reinsburgstraße"^^<xmls:string> ;
  <foaf:city> "Stuttgart"^^<xmls:string> ;
  <foaf:country> "germany"^^<xmls:string> .
```

Fig. 1, b) shows a *variant* of the UI for mobile devices. The structure remains the same but is rendered for a different target device. In addition it does not offer the possibility to book an open-yaw-flight to reduce the complexity of the application.

These examples show the non-trivial nature of UIs, including dynamic behaviour that is not inferable from simple data models: additional information is needed (e.g., rules for showing additional questions when input changes). Furthermore, the structure presented to the user for input differs from the structure of the actual request required by the backend service.

The goal of the presented approach is to provide a sharable way for describing UIs in a technology-agnostic manner for the automatic derivation of UIs for different contexts. Thus, the following requirements have to be considered:

- ***Req.1:*** A UI description is required, addressing the complexity of non-trivial UIs. It needs to contain all information about the data to be gathered and for the automatic generation of UI variants for different technologies and plattforms.
- ***Req.2:*** Information has to be provided, allowing the mapping of entered data to instances of the target ontology required by consuming services.
- ***Req.3:*** To achieve sharable UI descriptions, a non-proprietary description is required that con-

tains a minimum set of artefacts to be shared.

- ***Req.4:*** A process for (a) building final UIs and (b) for inferring instance data from user input that can be processed by (arbitrary) linked-data driven backend services.

3 PROPOSED SOLUTION

To meet the above requirements, we propose a ***single, declarative, data-centric application description***, which incorporates the required information (1) to derive non-trivial UIs and (2) for the mapping of input data to target ontology instances (cf. ***Req.1, Req.2***). To be applicable to multiple contexts, a *UI technology-agnostic model* is used, which is based on the data to be processed by the application. Here we base on previous work on data-centric UI description models proposed in (Hitz, 2016). This approach is applied to ontological concepts (Section 4.1) and extended to contain additional data, required for the mapping of input data onto target instances (Section 5).

To meet requirement ***Req.3,*** our solution uses RDF/OWL (Hitzler, 2009) ontologies. RDF/OWL is used, as it is a well understood, widely adapted technology, already applied to different contexts and for which tooling is available (e.g. reasoners, APIs). The result is a sharable **Application Ontology** (AO) containing the required information.

Fig. 2 shows a **solution scenario** for the use of the proposed Application Ontology in a linked data environment. The central elements are the *Target Ontology* (TO) and the corresponding *Application Ontologies* (AO) - both sharable between multiple client applications and backend services. The TO defines the semantics of possible input data for a business process. The AOs define variants of the user data to be gathered as outlined above. Fig. 2 shows the process for generating UIs based on the AO and the TO instance based on the input data (adressing ***Req.4***).

A generic *Client Application* selects an AO ❶ and generates the final UI to be displayed, using a *User Interface Transformation* based on the UI-related information available in the AO ❷. The resulting UI is integrated into the UI of the client (e.g., as a mesh-up component). It is presented to the user for input, based on the UI-related information of the AO (structure, dynamics, etc.).

When the user has entered data, it (i.e., an instance of the AO) is sent to a *Target Instance Transformation*, which exploits the mapping-related information available in the AO to generate an instance of the TO ❸. Since the TO instance is understood by the linked data service in the backend, it is can be used as input for the business process.

The approach has benefits for the automated generation of UIs for linked data applications:

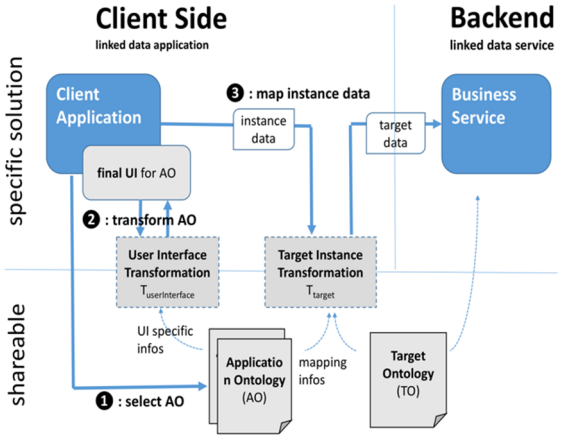


Figure 2: Solution scenario architecture.

- It uses a sharable artefact that allows generating UIs, that can be integrated into generic linked data applications
- The generated UIs are able to produce output for arbitrary linked data services by incorporating mapping rules for arbitrary target ontologies
- It uses a single, self-contained artefact to be easily shared and integrated into arbitrary applications

The following sections focus first on the information needed for UI derivation and its ontological description, then on the enhancement of that model regarding information needed to derive a target instance from user input. Finally, we outline the processes for UI derivation and target ontology instance generation.

4 APPLICATION ONTOLOGIES FOR UI DERIVATION

The proposed *Application Ontology* is partly based on results of previous work of the ***mimesis project*** (Hitz, 2016). The approach uses a model of the data processed by the application as foundation, which is enhanced by additional information regarding its semantics. The following sections summarize this information and show its application to an ontological application description.

4.1 Information Needs for Automatic UI Generation

To derive the information required for generating UIs, a set of interaction patterns was identified by analysing existing, frequently used 'real-life' applications along with an analysis of related work. Following, the data necessary to build UIs for these patterns was extracted. The summarized result is grouped into two categories (*Type related & Structural Information* / *Behavioural Information*). It is detailed in Table 1 along with the usage of the information within a UI derivation process.

Type related and Structural Information (I_1-I_4) describes data elements (i.e., types and type restrictions like *ranges* or *allowed values*), their structure (i.e., *grouping* and *hierarchical correlation*), and a meaningful *temporal sequence* of the questions.

Behavioural Information (I_5-I_7) models dynamic aspects of the UI at runtime. This includes conditions about the *existence / activation* of *elements / groups* bound to the content of other data elements within the model, the indication for complex *validations*, *operations* triggered on changes of the input data (*reactions*) or triggered by the user (*actions*).

Table 1: Information needs and usage for UIs (Hitz, 2016).

Ref	Information Need	Usage for UI Derivation
type related & structural information		
(I_1)	**type information for a data element or group** (based on XMLSchema)	selection of suitable input control based on type restrictions (e.g. presets and value ranges); provision of type-related validations
(I_2)	**hierarchical grouping of elements**	grouping of questions into display units; dependencies and hierarchical inclusion of groups; derivation of suitable navigation structures (sequential, tree, ...).
(I_3)	**temporal succession of data- or group elements**	display order of groups and input controls
(I_4)	**semantic cohesion of elements**	arrangement of controls (e.g. proximity of a *zip code* and *city*); identification of possible breakpoints for pagination
behavioural information		
(I_5)	**existence and activation conditions for data and group elements**	show/hide or de-/activate groups and questions, triggered on change of already entered data.
(I_6)	**validation operations**	trigger (complex) validations operations usually related to already entered data
(I_7)	**actions and reactions**	trigger operations on change of already entered data (reaction) or initiated by the user (action).

Based on the findings, a meta model can be created that incorporates the identified information and serves as a foundation to develop data descriptions for interview applications. Fig. 3 shows this meta model as UML diagram.

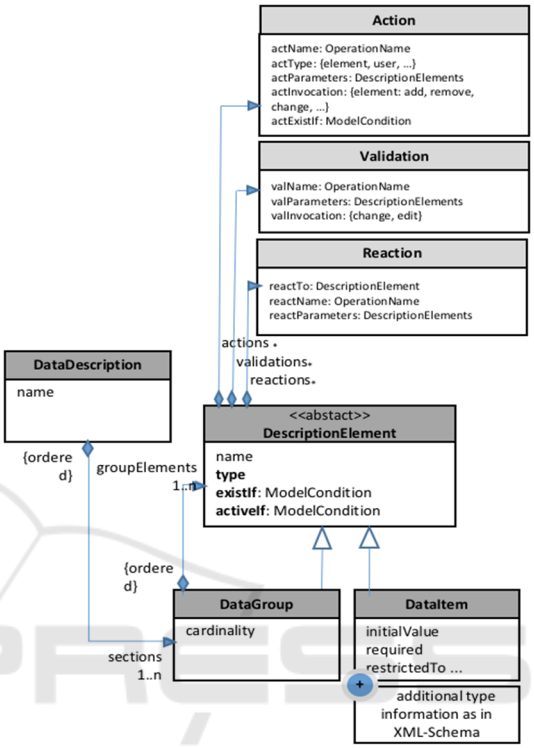


Figure 3: Meta-model in UML notation (Hitz, 2016).

A data description (*DataDescription*) consists of a succession of data groups (*DataGroup*) that might contain an ordered list of further groups or data elements (*DataItem*). This constellation allows to model the requested **structural information** regarding cohesion, (hierarchical) grouping and temporal sequence of the elements (I_2, I_3, I_4). Groups and data items are detailed by attributes / facets. E.g., **type information** (I_1) and **existential and activation conditions** (I_5) can be specified for each description element in the model. Further facets are used to specify the element more precisely in terms of data related aspects, i.e., *type restrictions* that are usually part of a type system like XML-Schema (I_1). Table 2 summarizes the semantics of the facets for *DataGroups* and *DataItems*. In addition, each description element might have associated **validation-, reaction- and action operations** (I_6, I_7), which are complemented by further facets like name of the operation, triggering events, and model elements required for the execution of the operation (cf. Hitz, 2016).

Table 2: Facets for DataGroups and DataItems.

Facet	*Description*	*Contents*
DescriptionElement		
name*	unique name as identifier for the element	[a-zA-Z0-9]+
type	*type of the group or data item*	s. below
existsIf	Condition for the existance of the group or element. if it evaluates to true, the data is relevant and presented	boolean expression. Referencing model items.
activeIf	Condition for the editability of the group or element. if it evaluates to true, the data is editable , else just displayed.	boolean expression. Referencing model items.
DataGroup		
type	type of the group	
cardinality	possible cardinality of the group. Defines, how often the group might be repeated. (e.g.used to express, that a person might have multiple addresses)	***:** no limit <n>: fixed value <n>..<m>: range
DataItem		
type	type of the data item **simple datatypes:** semantics according XML-Schema **custom datatypes** e.g. domian or context specific. implies additional behavior (e.g. country specific validation for a zip code).	**simple datatype:** text, number, boolean, date, float **custom datatype:** email, zipcode, phone, licenseplate
+ restrictions	additional type specific constraints XMLSchema (e.g. min/maxInclusive)	additional facets for datatypes
restrictedTo	restriction of possible vaules	Value ranges, e.g. dog|cat|mouse
+ multiple	allows multiple values to be selected	true, false
required	idndicates that the data is not optional	true, false
initialValue	initial value of the content	Depending on type and restrictions

The resulting model meets the requirements regarding the UI description (***Req.1***) as it permits a single description, containing all information to derive non-trivial UIs and, as it contains information about relations to model elements, allows consistency verification of the modelled UI.

4.2 Mapping to Ontologies

To get a sharable model, the meta-model is applied to RDF/OWL. The objective is to map the information requirements (I_1-I_7) listed in Section 4.1 towards RDF/OWL and hence develop a sharable **Application Ontology**. This is done by projecting the elements contained in the meta-model onto RDF/OWL elements.

Since ontologies in general are intended to describe entities, relationships, contained data elements and additional facts, expressing most of the structural information with RDF/OWL is straightforward: *DataGroups* can be modelled as *owl:Classes* and their hierarchical relations as *owl:ObjectProperties*. *DataItems* are defined as *owl:Data-typeProperties*.

To illustrate the mapping, Listing 2 shows an ex-

Listing 2: Application Ontology (excerpt) in OWL/Turtle notation.

```
@prefix : <http://mimesis/bookers/flight/v1#> .
@prefix : <http://mimesis/bookers/flight/v1#> .
@prefix mdt: <http://mimesis /datatypes#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>. ...
@prefix ma: <http://mimesis/annotations/v1>
@prefix sa: <http://mimesis/linkeddata/v1>
@base <http://mimesis/bookers/flight/v1> .

(1) #### Classes :
:Flightbooking rdf:type owl:Class .
:Basictraveldata rdf:type owl:Class .
:Flightinfo rdf:type owl:Class .
:Customerinfo rdf:type owl:Class .
:Persons rdf:type owl:Class .
:Flight rdf:type owl:Class .
:Returnflight rdf:type owl:Class .
:Openjawflightinfo rdf:type owl:Class .
:Customer rdf:type owl:Class .
:Address rdf:type owl:Class .
...

(2) #### Object Properties:
:Flightbooking.basictraveldata
      rdf:type owl:ObjectProperty ;
   rdfs:range :Basictraveldata ;
   rdfs:domain :Fligh tbooking .
: Flightbooking.flightinfo
      rdf:type owl:ObjectProperty ;
   rdfs:domain :Flightbooking ;
   rdfs:range :Flightinfo .
:Flightbooking.customerinfo
      rdf:type owl:ObjectProperty ;
   rdfs:range :Customerinfo ;
   rdfs:domain :Flightbooking .

:Basictraveldata.persons
      rdf:type owl:ObjectProperty ;
   rdfs:domain :Basictraveldata ;
   rdfs:range :Persons .
:Flightinfo.flight rdf:type owl:ObjectProperty ;
   rdfs:range :Flight ;
   rdfs:domain :Flightinfo .
:Flightinfo.returnflight
      rdf:type owl:ObjectProperty ;
   rdfs:domain :Flightinfo ;
   rdfs:range :Returnflight .
:Returnflight.openjawflightinfo
      rdf:type owl:ObjectProperty ;
   rdfs:range :Openjawflightinfo ;
   rdfs:domain :Returnflight

(3) #### Data Properties
:Flight.fromdestination
            rdf:type owl:DatatypeProperty ;
   rdfs:domain :Flight ;
   rdfs:range xsd:string .
:Flight.todestination rdf:type owl:DatatypeP...;
   rdfs:domain :Flight ;
   rdfs:range xsd:string .
:Flight.startdate rdf:type owl:DatatypeProperty ;
   rdfs:domain :Flight ;
   rdfs:range xsd:date .
:Flight.returnflight rdf:type owl: DatatypeProp...;
   rdfs:domain :Flight ;
   rdfs:range xsd:boolean .
:Returnflight.returndate rdf:type owl:Datat...;
   rdfs:domain :Returnflight ;
   rdfs:range xsd:date .
...

(4) #### UI Annotations
:Flightinfo.flight
   ma:sequence "1" ;
:Flight.startdate
   ma:sequence "1" ;
   ma:type "date" ;
:Flight.fromdestination
   ma:sequence "2" ;
   ma:restrictedTo "flightbooking
            .getDepartureAirports()" ;
   ma:type "text" ;
:Flight.todestination
   ma:restrictedTo " " ;
   ma:sequence "3" ;
   ma:activeIf "fromdestination.length0" ;
   ma:reactions "fromdestination:flightbooking
      .changeDestinations( fromdestination,
                  $todestination)" ;
   ma:type "text" ;
:Flightinfo.returnflight
   ma:existsIf "(returnflight == true)" ;
   ma:sequence "2" .
:Flight.returnflight
   ma:sequence "4" ;
   ma:type "boolean" ;
   ma:initialValue "true" .
:Returnflight.returndate
   ma:sequence "1" ;
   ma:type "date" ;
:Returnflight.openjawflightinfo
   ma:existsIf "(openjawflight == true)" ;
   ma:sequence "3" .
```

crept of the Application Ontology for the flight booking example introduced in Section 2. The *Classes* section (Listing 2, (1)) declares the *DataGroups* (e.g., *Flightbooking, Flightinfo, CustomerInfo*) as part of the application ontology (i.e., <http://...bookers/flight/v1#>). Examples for relations appear in the *Object Properties* section (e.g., *Flightinfo* as an object property of *Flightbooking* with range *flightinfo*). Contained *DataItems* appear in the *Data Properties* section (Listing 2, (3)) with information to which class they belong to, along with *basic* type information (e.g., exemplary data associated with a *Flight* and *ReturnFlight*). Using these basic RDF/OWL concepts, the structural information of I_2 and I_4 and partially I_1 are covered.

However, not all of the identified information can be expressed with standard RDF/OWL means. Since ontologies are intended for representing facts, they do not contain information such as the *sequence of data* (I_3)*, existential conditions* (I_5) or *functional aspects* (I_6, I_7). To the best of our knowledge, RDF/OWL does neither include a concept for the description of operations nor for declaratively modelling conditions / references based on instance data. To express this information, we use the **OWL annotation concept** as applied in (Khushraj et al., 2005) and (Gaulke et al., 2015) to produce a **profiled ontology.** This allows incorporating the information *declaratively* and leads to an ontology, that is (1) still covered by basic RDF/OWL (and thus can be used for standard reasoning) yet (2) exposes the additional information for reasoners (e.g., UI generators) that understand the specific profile.

Table 3: Additional annotations.

annotation	*content*	
type related & structural information		
:sequence	Number - position of the element in the flow of questions.	I_3
:type	Typeinformation for a group or element.	I_1
:<constraint>	typerelated constraints -> XMLSchema , e.g. **:restrictedTo, :initialValue, :max, :min**	I_1
behavioral information		
:existIf	Conditional expression References data within the hierarchy using path expressions at runtime for an instance.	I_5
:activeIf	Conditional expression References data within the hierarchy using path expressions at runtime for an instance.	I_5
:validations **:reactTo** **:actions**	Definition of validation, reaction and action operations **Validations syntax:** <trigger>:<operation>(<parameter >*) **Ractions syntax:** <element>:<operation>(<parameters>*) **Action syntax:** <type>:<trigger>:<operation>(<parameter>*)	I_6 I_7

Table 3 lists the used annotations of the proposed profile along with their mapping to the information needs. As an example, Listing 2 (4) shows annotations for *type, sequence, existence* and *reactions* applied to elements of the sample ontology.

The result is an ontological description of the UI-specific aspects of the application. It is sharable as an RDF/OWL ontology and thus meets requirements **Req.1** and **Req.3**. The mapping to RDF/OWL leads to an *ontological description* for dialog-based application UIs. It incorporates all information contained in the meta model of Section 4.1 so that UIs can be generated (cf. Section 6 and 7). The resulting UI is able to collect user input and provide it for further processing (i.e., as an instance of the AO).

Nevertheless, the approach has *limitations regarding its universality*. The consequence of using a *profiled ontology* with proprietary annotations is that a reasoner is required that is aware of the profile. The information is not interpretable by generic reasoners.

5 LINKING INSTANCE DATA TO TARGET ONTOLOGIES

To this point, the model does not contain information about how to provide the user input conforming the target ontology, understood by a linked data service. This section shows, how the data entered in the UI can be prepared for further processing.

When the user enters data, he actually builds an instance of the AO (*Application Ontology instance,* AOI). To produce a *Target Ontology instance* (TOI), a transformation from an AOI to a TOI is required. The main task is to build the required structure for the TOI and map data elements of the AOI into this structure. For the presented AOI, groups and data elements are related to elements of the TOI - although they might appear in a different structure. *DataGroups* are related to objects in the target ontology and *DataItems* to data properties of specific object instances of the TO.

Fig. 4 shows this for the flight booking example.

It shows that the *flightbooking* instance of the AOI is associated with the *:flightbookingrequest* of the TOI - as is the *flight* to the *:flight* instance. Flight information as the *startdate*, *from*- and *todestination* need to be mapped as data properties of the *:flight* instance. The *returndate* and additional open-yaw-flight information maps into the *:flight* instance, despite being part of a different *DataGroup* of the AOI (an example for structural differences between AO and TO).

Fig. 4 shows as well the information needed on the TOI side: to represent an object instance, its type needs to be known (e.g., *rdf:type Flight* for the *:flight* object instance). For a data property, its property name, type and the instance value is needed (cf. Listing 1, Section 2).

Since a requirement (***Req.2***) for the proposed Application Ontology is to contain all information to generate a TOI based on collected instance data, the mapping information needs to be integrated into the AO. To express this information, we use the **OWL annotation concept** as already applied in Section 4 for additional data semantics. Hence a (new) profile for expressing the linked data context is added for the AO.

The profile annotations used for that purpose within the AO are summarized in Table 4. For each *DataGroup* in the AO, that corresponds to an object instance in the TOI, an instance name (e.g. *:flight*) and the type needs to be specified. If the object is associated with another object (e.g. *:flight* as part of *:flightbookingrequest*, Fig. 4), information about the parent instance and the propertyname within that object needs to be supplied. For a DataItem, its type and propertyname is needed (e.g. *<fbo:startdate>* with type *<xmls:date>* for the departuredate of the flight, cf. Fig. 4) along with the instance, the dataproperty is associated with (e.g. *startdate* as part of *:flight*). Listing 3 shows the annotations for the flight example.

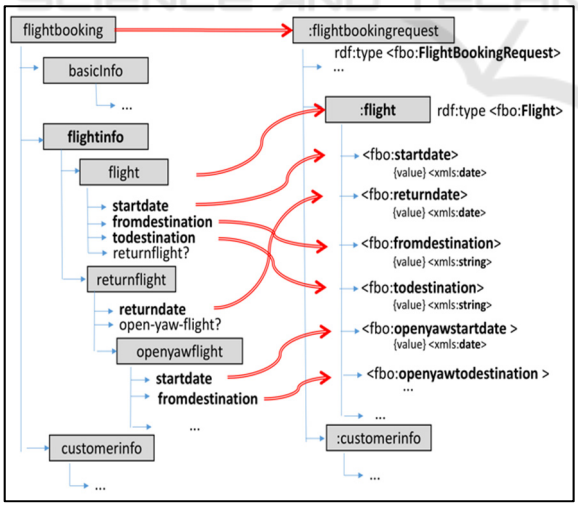


Figure 4: Mapping AOI data to TOI.

Given an AOI and the AO, the TOI can now be generated by tarversing the AOI tree nodes:

If passing a *GroupItem node* with annotated TO information, an RDF triple for an instance is created exploting the *DataGroup* annotations (cf. Table 4, *:swIndividual, :swClass*). If there is a relation to another instance, an ObjectProperty triple is generated to refelct the relation (*:swForIndividual, :swProperty*). If passing a *DataItem node*, a RDF triple for a DataProperty is created, using (a) the type, name and relation anntoations and (b) the instance data entered by the user for the corresponding field in the AOI.

Table 4: Linked data profile annotations.

Annotation	*Description*
DataGroup/ObjectProperty annotations	
:swIndividual	name of the instance to be generated
:swClass	object Type/Class of the group in target ontology
:swForIndividual*	name of the instance, this item is associated with
:swProperty*	object property name, this item has in the associated instance
* = for nested object properties only	
DataItem/DataProperty annotations	
:swType	type of the data property in the target ontology
:swForIndividual	name of the instance this data item is associated with
:swProperty	data property name, this item has in the associated instance

Listing 3: Linked Data Annotations.

```
:Flightbooking.flightinfo
   sa:swClass "fbo:FlightBookingRequest" ;
   sa:swIndividual "flightbookingrequest" .
:Flightinfo.flight
   sa:swClass "fbo:Flight" ;
   sa:swProperty "fbo:flight" ;
   sa:swIndividual "flight" ;
   sa:swForIndividual "flightbookingrequest" .
:Flight.startdate
   sa:swProperty "fbo:startdate" ;
   sa:swForIndividual "flight" ;
   sa:swType "xmls:date" .
:Flight.fromdestination
   sa:swProperty "fbo:fromdestination" ;
   sa:swForIndividual "flight" ;
   sa:swType "xmls:string" .
:Flight.todestination
   ... .
:Returnflight.returndate
   sa:swProperty "fbo:returndate" ;
   sa:swForIndividual "flight" ;
   sa:swType "xmls:date" .
:Openjawflightinfo.openyawfromdestination
   sa:swProperty "fbo:openyawfromdestination" ;
   sa:swForIndividual "flight" ;
   sa:swType "xmls:string" .
:Openjawflightinfo.departuredate
   sa:swProperty "fbo:openyawstartdate" ;
   sa:swForIndividual "flight" ;
   sa:swType "xmls:date" .
... .
```

This approach allows the automatic generation of a suitable TOI from an AOI based on information contained in the AO and thus meets ***Req.2*** and ***Req.3***.

Our approach uses a simplified method for mapping AO instance data to the TOI, allowing only a unidirectional mapping of the data onto the TOI. This restricts the contained data to the usecase we focus on, but does not allow mapping back from TOI to an AOI (for example, this could be used to preset data). There exists research on bidirectional tree transformations (e.g. Foster et al., 2005), which can be applied to extend the proposed solution in future work. Additionally, since a profiled ontology is used, the restrictions discussed in Section 4 also apply here.

6 GENERATING UI- AND TARGET ONTOLOGY INSTANCE

As outlined in Section 3, the two steps are the *User Interface Transformation* and the *Target Instance Transformation* building a Target Ontology instance when user input is ready. Fig. 5 summarizes the steps needed for the overall solution.

To generate a UI based on the AO, the approach presented by *mimesis* (Hitz, 2016) is used. It is based on the concepts of the CAMELEON framework (Calvary et al., 2002).

As shown in Table 1, the information contained in the AO is used for the derivation of UIs. Fig. 5 (on the left) outlines the different steps. The process starts with an instance of the *data-centric core model*, which is built from the information contained in the AO. The core model describes the processed data of the application according to the structure and properties presented in section 4.

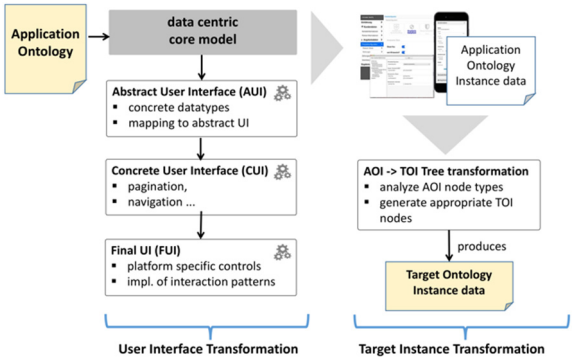


Figure 5: Derivation process.

Step 1: the core model is transformed to an *abstract UI* (AUI) using information about the *context of use* to concretize the information contained in the data-centric model. This step is crucial to generate usable UIs from a solely data-centric model that intentionally omits technical details. This includes enrichment with labels, explaining texts and help information (depending on the language), the mapping of data types to concrete types of the AUI (e.g., mapping zip to a text field restricted to 5 digits for Germany) and abstract UI input elements. The information needed here is derived from I_1, I_2, I_3 and I_4 (cf. Table 1)

Step 2: derives a *concrete UI* from the AUI description by incorporating the *device context* for which the UI is intended. It maps fields to pages by using information about device restrictions and exploits cohesion information contained in the data-centric model. The latter indicates how a flow of questions may be split up and positioned on pages for different device categories. The information needed here is derived from I_2 and I_4.

Step 3: Depending on the *technological context* the final UI is derived by generating now concrete UI Widgets for the abstract controls of the AUI and by implementing the functional aspects for the specific platform. This exploits the behavioural information contained in the basic application model. The information needed here is derived from I_1, I_5, I_6 and I_7.

These steps lead to a final UI, which can be run on a specific platform and presented to the user for input. When the user finishes his input, an AOI is available, based on the associated AO containing the input data. That now needs to be mapped to an instance of the TO. This was outlined already in Section 5 – resulting in the last, deferred step of the process.

Step 4: Traversal of the instance data tree within the AOI and generation of a TOI based on the instance mapping annotations contained in the AO. The resulting data object can be consumed by a linked data service following the target ontology.

7 VALIDATION

The following section focuses on the validation of the stated objectives to show, that (1) ontologies can be used to describe application UIs in a non-proprietary way, which (2) can be used to produce output conforming a target ontology and (3) are sharable within generic linked data applications.

The validation was carried out in association with a major German insurance company (Allianz Deutschland) from which we got data for the evaluation and which already uses parts of our implemen-

tation results in production environments (i.e., to generate UIs of *electronic risk acceptance check* applications for different products on customer and agent portals).

The company provided a set of typical 'real-life' dialog-based applications that were used during the analysis phase and the evaluation of the implementation. From this set, relevant applications were selected that cover the interaction patterns identified during analysis and to demonstrate the usefulness of the automated process and the Application Ontology developed in this paper.

To allow a deeper investigation, the following DOI (https://doi.org/10.13140/RG.2.2.32129.45929) is provided that lists sample resources for the *flightbooking application* used throughout this paper. It presents a working example of application variants and the complete application models.

Basic Setting and Preparation. We chose the architecture and building blocks outlined in Section 3 (cf. Fig. 2) and implemented the required components for that setting to be used in our validation steps.

First, the *User Interface Transformation* component was implemented as outlined in Section 6, which resulted in a UI *Transformation Service* (exposed as a web service). The implementation is based on available components from previous work (Hitz, 2016). We reused the transformation and – for a comparative evaluation – an import module for a proprietary application DSL (Domain Specific Language). The *UI Transformation Service* transforms a data-centric core model to a final UI for different platforms. It focuses on web-based dialog applications (using HTML, JavaScript, CSS) for different device categories (mobile, desktop).

As a second step, an import module was implemented, reading the proposed Application Ontology and converting it into the core data model of the *Transformation Service*.

Third, the *Target Instance Transformation* (cf. Fig. 2) was implemented as a web service. It consumes an AO and instance data as input, producing a TOI based on the contained data as outlined in Section 5.

Applicability of Ontologies. To validate the applicability of the ontological approach, a *comparative evaluation* was chosen based on the implementation of the UI *Transformation Service*. Fig. 6 shows the basic setting for the evaluation. The goal is to demonstrate that the proposed ontology has the same expressive power as the *mimesis* DSL, which was already evaluated in previous work. To achieve this, the same applications were modelled using (1) the *mimesis* DSL and (2) the Application Ontology. Both were transformed to the core model of the transformation service and the generated output was compared.

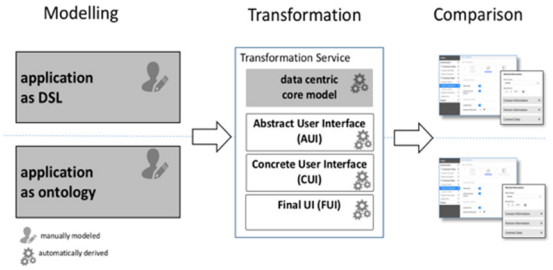


Figure 6: Basic setting for comparison.

Results. The results show that both kinds of descriptions can be mapped to the same core model and bear the same expressive power. The implementation shows that the proposed approach for using Application Ontologies to describe UIs leads to the same results as the solution using the proprietary *mimesis* DSL. While not being a formal proof, the results indicate that the data-centric approach may be applied to ontological descriptions of dialog applications. The evaluation showed as well, that the DSL (as proprietary approach) was much easier to use and less error prone than manually building AOs from scratch. But since the expressive power of both approaches is the same, it is possible to use the DSL for modelling and automatically transform the model into the proposed AO – preserving the benefits of both approaches.

Sharable, Reusable Application Descriptions. For the suitability of the proposed ontology as sharable, reusable application descriptions for linked data applications, we applied the approach to a concept for *Distributed Market Spaces* working with generic UIs for the specification of complex product requests. This concept is already published in (Hitz, Radonjic-Simic et al., 2016) and summarized here. The objective is to show that Application Ontologies can be (1) shared and used to generically build composed UIs and (2) can produce linked data requests – in this case to build a complex product request from user input.

Fig. 7 shows the basic architecture of the demonstrator. As generic user frontend a *Complex Product Builder* (CPB) application was implemented, that lets users search and select arbitrary *Application Ontologies* (AO) as proposed in this paper (Fig. 7, ❶). These were drawn from a shared UI description repository containing AOs for different product

components (e.g., booking a concert ticket or a flight). The user-selected AOs are sent to the *Transformation Service* (Fig. 7, ❷), which returns generated UIs for each AO. These were aggregated to a final UI (Fig. 7, right). Since the UIs are generated from the elements contained in the AO, the user input relates to the corresponding ontology elements. This allows building *an instance model* for each presented AO containing the input data of the user using an *Ontology Mapper* (Fig. 7, ❸). The result is a set of ontology instances on which a reasoner can build a *complex product request*, which is sent to the *Market Space* for further processing (i.e. generating a quote for the requested product components).

Results. Although the demonstrator is still a proof-of-concept it shows that sharing application descriptions is possible. In future work, AOs could be assembled from arbitrary sources (e.g., topic-related repositories for *insurance*, *travel planning,* etc.) and UIs for arbitrary domains can be generated. In addition, it shows that target ontology instances can be derived from user input data using the mapping information contained in the AO.

Although the approach lead to satisfying results and is easy to implement, we observed a drawback regarding the user friendliness in modelling the mapping for bigger AOs: hence the approach is focused on the AO, the information of the TOI is scattered all over the AO model and thus hard to grasp and maintain. Future work might focus on better tool support for this task or advanced mapping concepts.

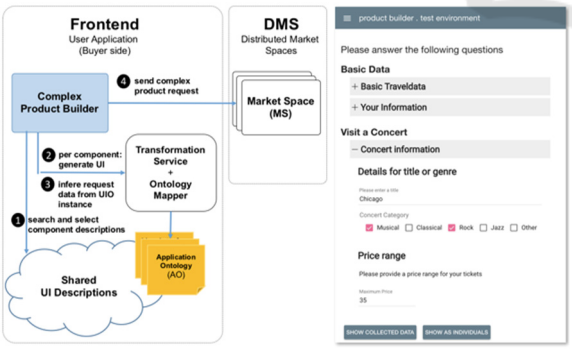


Figure 7: Generic UIs for complex product requests.

8 RELATED WORK

The research on the automatic generation of UIs covers many contributions during the last years that are based on model-driven concepts.

User Interface Description Languages (UIDL) focus mainly on the description of concrete UIs in a technology independent way. Examples are *JavaFX* (Fedortsova, 2014*)*, *UIML* (Abrams et al., 1999), *UsiXML* (Limbourg, 2004) and *XForms (W3C)*. The basic idea is to model dialogs and forms by using technology independent descriptions of in-/output controls and relations between elements (e.g. visibility) within a concrete UI. **Task-/conversation based approaches** describe applications by dialog flows which are derived from task models – e.g. *CAP3 (*Van den Bergh et al., 2011*)*, *MARIA* (Paterno et al., 2009) and conversation based approaches e.g. (Popp et al., 2009). They focus on a concrete model of the dialog flows. To generate an application frontend, the steps in a dialog flow are associated with technology independent UI descriptions displayed to the user. **Data-centric approaches** can be found in *JANUS* (Balzert te al., 1996) and *Mecano* (Puerta et al., 1994) which use a **domain model** as starting point for the derivation of UIs. While JANUS was designed to only provide CRUD-like interfaces for applications that work on a persisted domain model that does not support much dynamics in the UI, *Mecano* adds these aspects to its description.

Existing **Ontology based approaches** generally rely on the concepts of the mentioned approaches and use ontologies to represent the information about concrete UIs. For instance, in analogy of UIDL approaches, Liu et al. (2005) propose an ontology driven framework to describe UIs based on concepts stored in a knowledge base. Khushraj et al. (2005) uses web service descriptions to derive UI descriptions based on a UI ontology, adding UI related information to the concept descriptions (profile). In analogy with task based approaches, Gaulke et al. (2015) use a profiled domain model enriched with UI related data to describe a UI and associate it with an ontology driven task model.

Dissociation: A main goal of the proposed approach was to minimize the number of needed artefacts and to use a sharable representation that can be reused in different contexts. The models of the aforementioned approaches usually do not contain enough semantical information for reasoning that could be used for deriving UI variants. The UIs are manually modelled using a large amount of artefacts. This opens a gap in automating the process for building UIs. In addition, the produced artefacts are usually proprietary and UI-specific.

The solution proposed in this paper is based on the application's processed data and enriches its model by additional semantics. This leads to a **single, central description for the application that**

serves as a knowledge base for the automatic derivation of UI variants. The data-centric approach allows the reuse of the model in different contexts and - by using a non-proprietary representation for the model - the sharing and integration into different environments.

9 CONCLUSIONS

In this paper a model-driven approach for the automatic generation of UIs for dialog-based linked data applications is presented. It is based on an UI-agnostic, ontological model of the processed application data enhanced by type-related, structural and behavioural information to generate non-trivial UIs. Additionally, it contains information on how input data maps to linked data input of target business services – enabling the generated UIs to be used in a linked data services ecosystem.

In the course of the paper, the information needs are identified and a meta-model is derived from which non-trivial UIs can be inferred. The information needs are mapped to an ontological description, relying on RDF/OWL constructs to get a non-proprietary representation. The mapping of input data to target ontology instances is shown and the process to derive UIs and target data is outlined. Finally, the evaluation is presented which provides an implementation of the generation process for UIs from an Application Ontology.

The results of the evaluation indicate the feasibility of the proposed Application Ontology to be used for generating UIs for dialog based linked data applications. Since the number of artefacts is reduced to a single, UI-agnostic application model, containing information for UI generation and produce an outcome understood by linked data services, the manual step for building UIs can be eliminated. Using a universal representation as RDF/OWL allows the application model to be sharable and the contained semantics can be exploited using standard tools for reasoning on the model and instances.

The approach is intentionally limited to dialog based, interview-like applications, that are very important and frequently used in enterprise information systems (e.g., in the insurance domain). Since a limited set of applications was used for analysis, we do not claim completeness of the identified interaction patterns. The practical use of the approach might bring forth additional interaction patterns, extending the basic information set in future. Regarding the proposed use of ontologies, the evaluation strongly indicates the usefulness for UI derivation – though it uses proprietary annotations and thus restricting its universality. Future work might concentrate on finding more general ways for incorporating the information.

REFERENCES

Abrams, M. et al., 1999. UIML: An appliance-independent XML user interface language. In *WWW '99 Proceedings of the eighth international conference on World Wide Web*. pp. 1695–1708.

Balzert, H., Hofmann, F. & Kruschinski, V., 1996. The JANUS Application Development Environment - Generating More than the User Interface. In *Computer Aided Design of User Interfaces*, Vol. 96. pp. 183–206.

Calvary, G. et al., 2002. The CAMELEON Reference Framework.

Coutaz, J., 2010. User interface plasticity: model driven engineering to the limit! In *EICS '10 Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*. pp. 1–8.

Fedortsova, I. & Brown, G., 2014. JavaFX Mastering FXML, Release 8. JavaFX Documentation. Available at: http://docs.oracle.com/javase/8/javafx/fxml-tutorial/preface.htm.

Foster, J., Greenwald, M. & Moore, J., 2005. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. *ACM SIGPLAN*, 3, pp.1–64.

Gaulke, W. & Ziegler, J., 2015. Using profiled ontologies to leverage model driven user interface generation. *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '15*, pp.254–259.

Hitz, M., 2016. mimesis: Ein datenzentrierter Ansatz zur Modellierung von Varianten für Interview-Anwendungen. In V. Nissen et al., eds. *Proceedings - Multikonferenz Wirtschaftsinformatik (MKWI) 2016*. pp. 1155–1165.

Hitz, M. et al., 2016. Generic UIs for requesting complex products within Distributed Market Spaces in the Internet of Everything. In F. Buccafurri, ed. *Proceedings of CD-ARES 2016, LNCS 9817*. F. Buccafurri et al.

Hitzler, P. et al., 2009. OWL 2 Web Ontology Language Primer. W3.org. Available at: http://www.w3.org/TR/2009 /REC-owl2-primer-20091027/.

Khushraj, D. & Lassila, O., 2005. Ontological approach to generating personalized user interfaces for web services. *The Semantic Web–ISWC 2005*, pp.916–927.

Kraus, A., Knapp, A. & Koch, N., 2003. Model-Driven Generation of Web Applications in UWE. *Proc. 3rd Int. Wsh. Model-Driven Web Engineering (MDWE'07). CEUR-WS 261*

Limbourg, Q., 2004. USIXML: A User Interface Description Language Supporting Multiple Levels of Independence. In M. Matera & S. Comai, eds. *ICWE Workshops*. Rinton Press, pp. 325–338.

Liu, B., Chen, H. & He, W., 2005. Deriving user interface from ontologies: A model-based approach. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI, 2005*, pp.254–259.

Meixner, G., Paternò, F. & Vanderdonckt, J., 2011. Past, Present, and Future of Model-Based User Interface Development. *i-com*, (3), pp.2–11.

Paterno, F., Santoro, C. & Spano, L.D., 2009. Maria: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environment. *ACM Transactions on Computer-Human Interaction*, 16(4).

Pfisterer, D., Radonjic-Simic, M. & Reichwald, J., 2016. Business Model Design and Architecture for the Internet of Everything. *Journal of Sensor and Actuator Networks*, 5(2), p.7.

Popp, R. et al., 2009. Automatic generation of the behavior of a user interface from a high-level discourse model. In *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences, HICSS*.

Puerta, A.R., Eriksson, H., Gennari, J.H., Musen, M.A.,1994. Beyond data models for automated user interface generation. *Proceedings British HCI'94*.

Van den Bergh, J., Luyten, K. & Coninx, K., 2011. CAP3: Context-Sensitive Abstract User Interface Specification. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems - EICS '11*. pp. 31–40.