Using Application Ontologies for the Automatic Generation of User Interfaces for Dialog-Based Applications

Michael Hitz, Thomas Kessel

▶ **To cite this version:**

Michael Hitz, Thomas Kessel. Using Application Ontologies for the Automatic Generation of User Interfaces for Dialog-Based Applications. 10th International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS), Dec 2016, Vienna, Austria. pp.16-31, 10.1007/978-3-319-49944-4_2 . hal-01630537

HAL Id: hal-01630537
https://inria.hal.science/hal-01630537

Submitted on 7 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Using Application Ontologies for the Automatic Generation of User Interfaces for Dialog-Based Applications

Michael Hitz[1], Thomas Kessel[1]

[1] Cooperative State University Baden-Wuerttemberg, Stuttgart, Germany
{michael.hitz, thomas.kessel}@dhbw-stuttgart.de

Abstract. The paper presents a data-centric, model driven approach for the automatic generation of user interfaces (UIs) for dialog-based applications using ontological descriptions. It focuses on Interview Applications, a common pattern e.g. for self-service applications in EIS. Existing approaches for the automatic UI generation usually rely on proprietary, UI-specific description models, designed and developed manually for the application in focus. The manual creation of the artefacts leads to a gap in the automated development, i.e for dialog-based application UIs, where the structure and behavior is driven by the processed data. Furthermore, the UI specific nature of the artefacts impedes their (re-)use in different contexts. The presented approach is a shift away *from a UI-specific towards a data-centric* method of modelling dialog-based applications, bridging this gap. Application-Ontologies are used as description means, which leads to reusable, sharable model artifacts, applicable to different contexts of use.

1 Introduction

The ongoing digitalization of business processes in Enterprise Information Systems (EIS) raised the need for exposing different variants of User Interfaces (UI) to let different user groups interact with the systems in different contexts of use and supporting different platforms (e.g. as a desktop, mobile and web application for customers or insurance brokers).

A commonly observed pattern in sales related dialog-based applications is to collect data needed for the execution of a business process in a directed dialog in form of an interview (a.k.a form filling or directed dialog [4]). Examples for these **Interview Applications** can be found i.e. on the internet or web based business portals: e.g. the booking of a flight, the money transfer in a banking portal or the request of a quote for an insurance product. The application type is characterized by a high degree of standardization and clearly defined interaction concepts (mostly motivated by company style guides or platform standards). Thus the UIs for this application type are well suited for automatic generation.

Although there exist quite a lot of approaches for the model driven generation of UIs (see Related Work), they are not widely used in practice [14]. Mostly they rely on **manually modelling UI specific artefacts** (e.g. concrete UI descriptions, taskflow- and related data models), which are closely coupled and thus complex to create and

maintain. In addition, the UI specific nature and proprietary descriptions impedes their reuse in different contexts [6]. Furthermore, the manual development of the artefacts leads to a clear break and a gap in the automated application development – especially for data-driven applications as the aforementioned Interview Applications, where the structure and behavior of the UI is closely related to the data.

The objectives of this paper are to present an approach, that (1) bridges this gap by pushing the description from proprietary UI-specific information **towards a data-centric model** for applications and (2) to use common **non-proprietary descriptions** (i.e. RDF/OWL Ontologies) as modelling means.

The approach is based on the thesis, that structure and behavior of UIs for Interview Applications rely on the characteristics and semantics of the processed data. Thus UIs could be derived from data descriptions that contain the relevant semantic information. The proposed solution uses a *single, declarative model* of the processed data, which is *augmented by additional semantical information* used to infer all necessary information required to derive UIs.

The mayor benefits of this approach are to get (1) a **single, UI agnostic artefact** related to the processed data that allows an improved automation for application variant development and (2) a **non-proprietary, shareable application description**, applicable to different contexts of use.

The proposed approach in this paper contributes to the field of model driven development of user interfaces for dialog based applications. It adds concepts for the use of semantic knowledge about the processed data, its representation as ontology and its use to derive UIs

The paper is organized as follows: first the basic idea of a data-centric description, the character of Interview Applications and the information required to automatically generate user interfaces are elaborated. Then the representation of the information by means of ontologies and its concepts is presented followed by an outline of the derivation process for the UIs. Finally, evaluation of the concept and related work is presented, followed by a conclusion.

2 Data-centric Description of Interview Applications

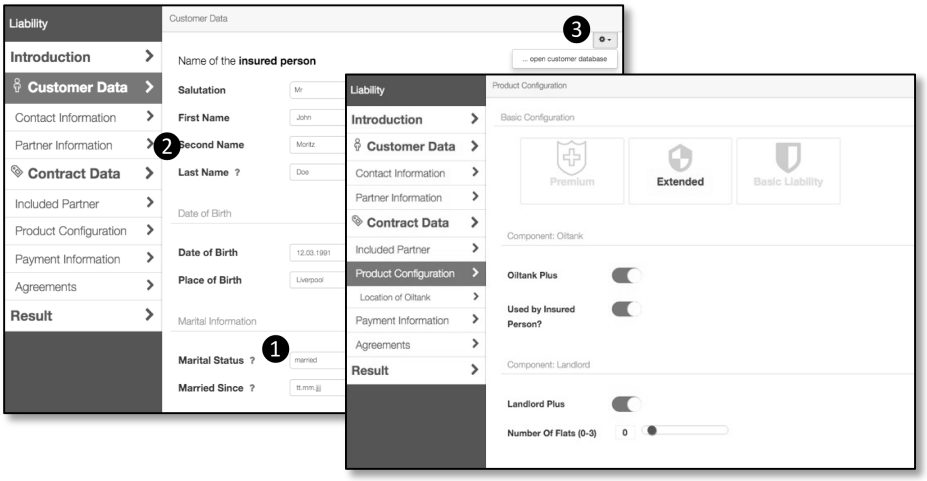
The basic assumption of the data-centric approach is, that manually developed UIs for Interview Applications *are built on the characteristics and semantics of the data processed by the application*. Application developers use this knowledge to build suitable frontends for the data – e.g. a *reasonable* selection, grouping and sequence of input elements, the showing/hiding of sections or the navigation between pages [6]. The knowledge is used implicitly by the developer and based on his experience or other rules, that are *tacit knowledge*. The basic idea is to incorporate this semantical knowledge into a data-centric model along with the processed application data.

The following sections illustrate the character of Interview Applications and show which information is needed to derive a UI.

2.1 Character of Interview Applications

Interview Applications *collect related data* in a *meaningful sequential flow of questions* in a *dialog with the user*. Depending on *already entered information the flow might change* and, if applicable, further questions are asked or omitted when necessary. The following example illustrates the data-driven character of Interview Applications. It contains most characteristics that need to be modeled in a data-centric description, listed in the next section.

Example: *quote for a liability insurance*. The computation of a quote for a liability insurance is chosen as a sample use case. Such calculators are a very common application type in the insurance domain and incorporate multiple interaction patterns common for Interview Applications.



a) Customer data b) Product Configuaration

Figure 1: UI for calculating a quote

Fig. 1 shows an example of a graphical UI as used by an agent. The agent enters successively data that is needed for computing a quote. In Fig. 1a) subsequent questions are asked concerning the customer (e.g. *name* and marital *status*). On the left side there is a hierarchical navigation that allows random switching between various question groups (e.g. *customer* and *contract data*).

The data input elements are chosen based on **type related properties** (e.g. the basic type, value ranges, restrictions etc.). They are **ordered in a semantically meaningful** manner (e.g. *name* information before *marital status*) and have a **hierarchical relation** to each other (e.g. on the left hand navigation of Fig. 1a) *contact information* is shown as part of *customer data*).

The processed data elements are semantically interrelated. This reaches from **related content** (e.g. the *zip code* is related to a certain *city*) to **existential relations**, e.g. the *date of marriage* and *partner data* only exist if the *marital status* is set to *married* (Fig. 1a, ❶❷). Additionally, dialogs show dynamical behavior: input is **validated** and field content might need adjustment as a **reaction to changes** in other fields (e.g. prefilling

the *city* according to a given *zip code* or explicitly initiated **user actions** (e.g. opening a customer database to prefill customer data, Fig. 1a, ❸).

2.2 Information Needs for Automatic Generation of UIs

In previous work (cf. [8],[9]) we derived a set of interaction patterns and extracted information, that is needed to build UIs that behave as in the example above. This was done by analyzing existing 'real-life' applications used in a major insurance company and match the findings to existing work within the field of UI generation (e.g. [4][20]). The analysis leads to a set of information, needed to automatically derive UIs using these patterns. The information can be grouped into two categories (cf. Table 1).

Table 1. Information needs for a datamodel and its use to derive UIs.

ref	information need	usage for UI derivation
type related & structural information		
(I_1)	**type information for a data element or group** (e.g. based on XMLSchema)	selection of suitable input control based on type and specified restrictions (e.g. presets and value ranges); providing 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 within the flow of questions**	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
behavioral 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).

Type related and Structural Information (I_1-I_4):
This information is needed to describe the 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 succession* of the questions within the interview [5], which is based on the *semantical cohesion*.

Behavioral Information (I_5-I_7):
This is needed to model the dynamic, data-related aspects of the UI for examination at runtime; i.e. conditions about the *existence/activation* of *elements/groups* bound to the content of other data elements within the model, the indication for complex *validation*, *operations* associated with data elements and groups triggered on changes of the input data (*reactions*) or triggered by the user (*actions*) [15][20].

This set of information was found adequate to derive different aspects of the UI following the interaction patterns in focus. Table 1 summarizes the usage of the information within a derivation process which will be detailed in section 4.

Based on the findings, a meta model was developed that incorporates the identified information and served as a foundation to develop data descriptions for Interview Applications. Fig. 2 shows this meta model as UML diagram.

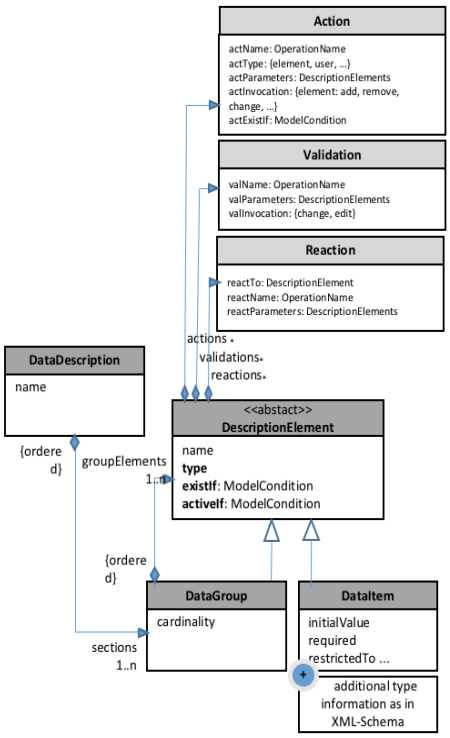


Figure 2: Metamodel in UML notation

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

Table 2: Facets for DataGroups and DataItems

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 succession (order) of the elements (I_2, I_3, I_4). Groups and data items are detailed by attributes/facets. E.g. the **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*. Additionally, each description element might have associated **validation-, reaction- and action operations** (I_6, I_7) . These are detailed by further facets (cf. Figure 2) like a *name* for the operation, *triggering events* and *references to model elements* needed for the execution of the operation (input/output parameters) [8]. Elements are referenced by an identifier in dotted notation, pointing out their position in the model hierarchy to be retrieved at runtime (cf. section 4, step3).

The resulting model addresses the first objective of this paper stated in section 1: an approach describing Interview Applications based on their processed data, augmented by data related information, that can be used to derive UIs. To achieve the second goal – using sharable and common means for the description – we apply this model to RDF/OWL as a common language in the field of semantic web technologies.

3 Using Ontologies as Application Description

The objective is to describe the mapping of the information requirements (I_1- I_7) listed in section 2.2. towards RDF/OWL and hence develop an **application ontology**. RDF/OWL [11] and its basic features are selected as a well understood, widely adapted technology used in different contexts for which tooling is available (e.g. reasoners, APIs). Table 3 summarizes the RDF/OWL features and concepts that are used to model the identified information requirements (I_1- I_7).

Table 3. Mapping of information needs to RDF/OWL features

ref	*Information needed*	*OWL feature mapping*
type related & structural information		
(I_1)	type information for an element / group	`OWLDataProperty`, `OWLAnnotations` (for additional details)
(I_2)	hierarchical grouping of elements	`OWLClass` `OWLObjectProperties`
(I_3)	temporal succession / flow of questions	`OWLAnnotation`
(I_4)	semantic cohesion of elements	`OWLClass`
behavioral information		
(I_5)	existence and activation conditions	`OWLAnnotation`
(I_6)	validation operations	`OWLAnnotation`
(I_7)	actions and reactions	`OWLAnnotation`

Ontologies in general are intended to describe entities, relationships, contained data elements and additional facts in a way that inferences are built upon that knowledge. Hence the mapping of most of the *structural information* as identified in section 2.2 to RDF/OWL is a straight forward task.

To illustrate the mapping, Listing 1 shows a simplified application ontology for the *customer data* example in section 1.1[1]: *DataGroups* are modeled as *owl:Class*es within an ontology and their hierarchical relations as *owl:ObjectProperties* (e.g. *customerdata* as an object property of a *Liability* with range *Customerdata*). The *Classes* section declares the *DataGroups* (e.g. *Customerdata*, *Contractdata* and *Fullname*) as part of the application ontology (i.e. an ontology for a liability insurance *<http://../liability/v1#>*). *DataItems* are defined likewise as *owl:DatatypeProperties*, containing information to which class they belong to, along with basic type information (e.g. Listing 1, exemplary data associated with an *Address)*. Using these basic concepts, the structural information of I_2 and I_4 and partially I_1 are covered.

[1] Due to the space restrictions of a paper, a complete example can be reviewed at https://doi.org/10.13140/RG.2.2.16564.24963

However not all of the identified information needed for UI generation can be expressed out-of-the-box. Ontologies are made for knowledge representation and therefore RDF/OWL does not contain information like *sequence of data* (I_3), *existential conditions* (I_5) or *functional aspects* (I_6, I_7) in its basic language. To the best of our knowledge, RDF/OWL does neither include a concept for the description of operations nor for declaratively modelling conditions/references on instance data. To express the information needed, we use the OWL annotation concept as used by [12][7] to produce a **profiled ontology.** This allows to incorporate the information *declaratively* and leads to an application 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 do understand the profile.

```
@prefix : <http://mimesis/insurance/liability/v1#> .
@prefix mdt: <http://mimesis /datatypes#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>. ...
@base <http://mimesis /insurance/liability/v1> .
#### Classes :
:Liability rdf:type owl:Class .
:Customerdata rdf:type owl:Class .
:Fullname rdf:type owl:Class .
:Maritalinformation rdf:type owl:Class .
:Address rdf:type owl:Class .
:Partnerinformation rdf:type owl:Class .
:Contractdata rdf:type owl:Class .
...
#### Object Properties:
:Liability.customerdata rdf:type owl:ObjectProperty... ;
    rdfs:domain :Liability ;
    rdfs:range :Customerdata ;
:Liability.contractdata rdf:type owl:ObjectProperty ... ;
    rdfs:domain :Liability ;
    rdfs:range :Contractdata ;
:Customerdata.address rdf:type owl:ObjectProperty... ;
    rdfs:domain :Customerdata ;
    rdfs:range :Address ;
:Customerdata.fullname rdf:type owl:ObjectProperty ... ;
    rdfs:domain :Customerdata ;
    rdfs:range :Fullname ;
:Customerdata.maritalinformation rdf:type
                        owl:ObjectProperty , ... ;
    rdfs:domain :Customerdata ;
    rdfs:range :Maritalinformation;
:Customerdata.partnerinformation rdf:type
                        owl:ObjectProperty;
    rdfs:domain :Customerdata ;
    rdfs:range :Partnerinformation ;
```

```
#### Data properties
:Address.building_no rdf:type owl:DatatypeProperty...;
      rdfs:domain :Address ;
      rdfs:range mdt:buildingNo ;
:Address.city rdf:type owl:DatatypeProperty .. ;
    rdfs:domain :Address ;
    rdfs:range mdt:text ;
:Address.country rdf:type owl:DatatypeProperty...;
    rdfs:domain :Address ;
    rdfs:range mdt:country ;
:Address.eMail rdf:type owl:DatatypeProperty... ;
    rdfs:domain :Address ;
    rdfs:range mdt:eMail ;
:Address.street rdf:type owl:DatatypeProperty... ;
    rdfs:domain :Address ;
    rdfs:range mdt:text ;
:Address.zip rdf:type owl:DatatypeProperty... ;
    rdfs:domain :Address ;
    rdfs:range mdt:zip ;

 Examples for Annotations
# sequence and additional type information
:Maritalinformation.status rdf:type ...    ;
    ma:initialValue "married" ;
    ma:restrictedTo "married|notmarried|divorced" ;
    ma:sequence "1" .
# existance
:Customerdata.partnerinformation rdf:type ...    ;
    ma:existsIf "(liability.customerdata
                .maritalinformation.status == 'married')".
# validations, reactions, actions
:Address.city rdf:type owl:DatatypeProperty    ;
    ma:sequence "4" ;
    ma:validations "change:validators.validateCity(
                liability.customerdata.address.zip)" .
```

Listing 1. Excerpt of the resulting RDF/OWL model in Turtle notation

Table 4 lists the used annotations within the proposed profile along with their mapping to the information needs. As an example, Listing 1 shows annotations for *type*, *sequence*, *validation* and *reactions* applied to elements of the sample ontology.

The proposed mapping onto RDF/OWL constructs addresses the second objective of this paper stated in section 1: it leads to an *ontological description* for Interview Applications. Hence it incorporates all the information contained in the meta model in section 2.2, UIs may be derived based on such an ontology (cf. section 4 and 5). Hence it uses a common language and describes the processed data for an application,

it can be used in different contexts and is not limited to UI generation. An example for a non-UI use will be given in the evaluation section.

Nevertheless, the approach has *limitations regarding its universality*. The consequence of a *profiled ontology* using proprietary annotations is, there has to be a reasoner that is aware of the profile. The contained information is not interpretable to general reasoners and thus it is not shared as 'world knowledge'. The proposed solution is consciously limited to *hierarchical ontologies*. This is not a restriction for Interview Applications as they operate on hierarchical data structures by definition. But this characteristic prevents the approach to be applied to arbitrary ontologies that might have a *reticular graph structure*. Sahar et al. [21] address this problem in the context of UI generation. The results found there may be used to extend the applicability of the proposed approach in future work.

Table 4. Elements of the annotation profile

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)

4 Derivation Process for UIs

As outlined in Table 1 (section 2.2), the information contained in the proposed model is used for the automatic derivation of UIs. Fig. 3 outlines the derivation process. The basic approach is based on the concepts of the CAMELEON framework as proposed by Calvary [3]. The starting point for the UI derivation process is an instance of the data-centric application model (*data-centric core model*). It contains the description of the processed data of the application according to the structure and properties presented in section 2.2.

Step 1: the core model is transformed into 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 phase includes the enrichment with labels, explaining texts and help information (depending on the language context), the mapping of data types to concrete types of the AUI (e.g. the mapping of the custom type *zip* to a *text field* restricted to 5 digits, if the language context is *German*) and abstract UI input elements. For instance, a *number range control* for a numerical value

having *min /max restrictions* or a *oneOfManySelection control* for elements *restricted to* a set of possible values. The information needed here is derived from I_1, I_2, I_3 and I_4.

Step 2: derives a *concrete UI* (CUI) from the AUI description by incorporating the *device context* for which the UI is intended. It includes the mapping of fields onto pages (pagination) by using information about device restrictions (e.g. for mobile devices) and exploiting the 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.

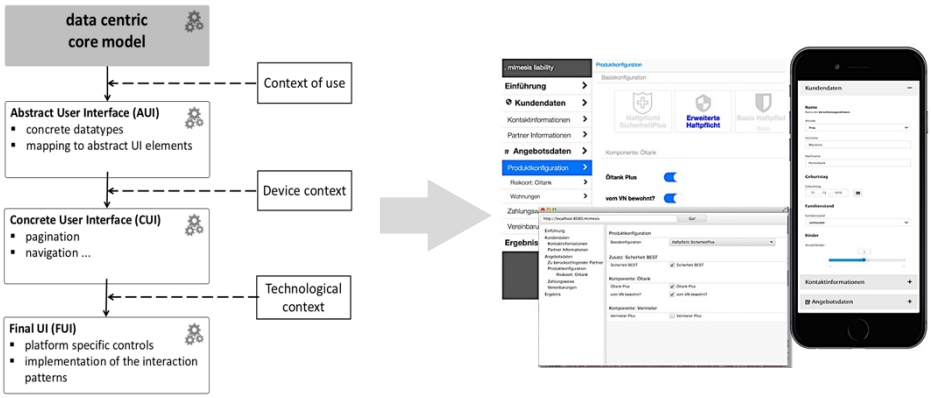


Figure 3: Derivation process for UI variants

Step 3: Depending on the *technological context* a final UI is derived by generating concrete UI Widgets for the AUI controls. Besides, an access mechanism to user entered data at runtime needs to be supplied, allowing the implementation of the functional aspects, e.g. a model for evaluation of visibility in Model-View-Controller application. The information for the functional aspects is derived from I_1, I_5, I_6 and I_7.

5 Demonstration and Evaluation

The following sections focus on the validation of the stated objectives to show that (1) a data-centric approach may lead to an increased automation and is suitable for generating UIs for Interview Applications, (2) ontologies can be used within the approach to describe application UIs in a non-proprietary way that are (3) shareable and thus applicable for different contexts of use.

Our research on the topic of data-centric application descriptions is conducted using the Design Science Research (DSR)[18] and Action Design [22] approach. The resulting artefacts (i.e. data-centric meta model, derivation process and ontological description) are refined in several iterations and evaluated per iteration by implementation and technical experiments with prototypes – which is a commonly applied technique for evaluation of algorithms and models [17].

The evaluation of the approach was conducted in association with a major German insurance company (Allianz Deutschland AG) from which we drew the data for the evaluation. The insurance company provided a set of typical ‘real-life’ Interview

Applications that were used for the analysis phase and the validation 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 afterwards the ontology developed in this paper.

To allow a deeper investigation, an online-link[2] is provided that lists sample resources for the *liability quote application* used throughout this paper. It shows a working example of application variants and the complete application models mentioned below.

Evaluation of the viability of the data-centric approach (1)

First, the derivation process outlined in section 4 was implemented resulting in a *Transformation Service* (exposed as RESTful webservice), which transforms a data-centric application description to a final UI for different platforms. The implementation was based on available components from previous work done by Hitz [8]. The implementation focused on web-based Interview Applications covering HTML/JavaScript UIs for different device categories (mobile, desktop). In addition, another prototype for rich client UIs using JavaFX was established recently.

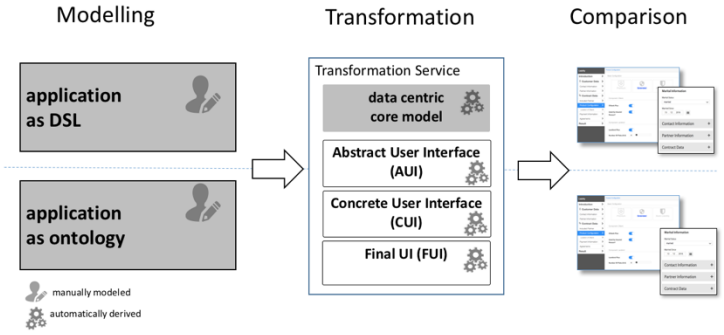


Figure 4: Basic setting for evaluation

Fig. 4 shows the basic setting for the implementation. The aforementioned selected applications were described as a first step by using a DSL (domain specific language) as proposed in [8] (Fig. 4, upper left), which contains a model following the proposed meta model (cf. section 2.2). These models were imported into the data-centric core model and used by the transformation service to produce final UIs for different platforms as outlined in section 4.

Results: The implementation of the transformation process and its application to existing Interview Applications showed that the information outlined in section 2.2 is appropriate to derive non-trivial UIs for the identified interaction patterns in practice. The functionality of the generated UIs corresponds to a large extent to the already existing manually designed counterparts which served as a basis for the analysis. It could be demonstrated, that a single artefact is sufficient to model Interview Applications and that the data-centric approach leads to a high degree of automation for generating different variants for the applications (e.g. different technologies, navigation, input styles).

[2] Link to website with additional content: https://doi.org/10.13140/RG.2.2.16564.24963

However, limitations could be observed in situations, where the outlined model did not contain *enough semantical information* for a selection of sophisticated widgets for the generation of the final UI. For example, the use of a *selection panel* using buttons instead of a *dropdown box* depends on the character of the question (like 'product component selection'). This issue was solved by extending the model with additional properties like *semantical tags* for an element.

The results of this implementation are already used in production environments of Allianz Deutschland AG, e.g. to dynamically generate the UIs of complex *electronic risk acceptance check* applications for different products on customer and agent portals.

Applicability of Ontologies (2)

To evaluate the applicability of the approach to ontologies, a *comparative evaluation* was chosen based on the implementation of the first step (Fig. 4). The goal was to demonstrate that the proposed ontology has the same expressive power as the DSL used in the first step and thus produces the same output. To achieve this, the same applications were modeled using the proposed ontology (cf. section 3) and an import module was implemented, that mapped the ontology contents to the core model of the transformation service (Fig. 4, lower left). This was used to generate final UIs, which were compared to the ones generated in the first step.

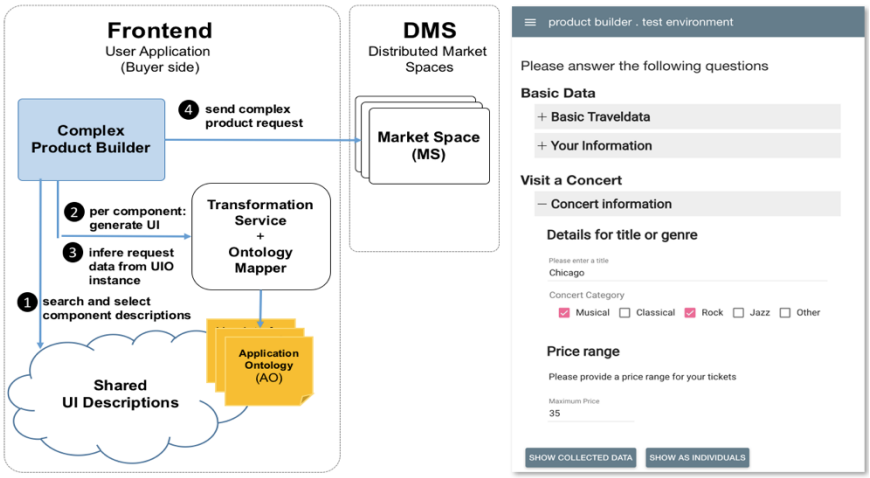
Results: The results clearly show that both kinds of description can be mapped to the same core model and bear the same expressive power. The implementation showed, that the proposed approach for using ontologies to describe Interview Applications leads to the same results as the solution using the proprietary DSL used in [8]. Though it is no formal proof, the result clearly indicates that the data-centric approach may be applied to ontological descriptions of Interview Applications.

Ontology Based, Shareable and Reusable Application Descriptions (3)

For the suitability of the proposed ontology as shareable, reusable application descriptions, we applied the approach to a concept for distributed marketspaces working with generic UIs for the specification of complex products. This work is already published in [10] and thus summarized here. The objective was to show, that application ontologies as proposed above can be (1) shared and used to generically build composed UIs and (2) can be used for non-UI-specific purposes – in this case to deduce a complex product request from the user input, that is an instance of the applications data model represented by the ontology.

For this purpose, a demonstrator was implemented that used the aforementioned results. Fig. 5a shows the basic architecture of the demonstrator. As generic user frontend a *Complex Product Builder* (CPB) application was implemented, that let the user search and select Application Ontologies (AO) as proposed in this paper (Fig. 5a, ❶). These are drawn from a shared UI description repository containing arbitrary UIOs for different product components (e.g. the booking of a *concert* or *flight*). The user selected AOs for his demand are sent to the *Transformation Service* (Fig. 5, ❷), which returns the generated UI partials for each AO. These are aggregated into a UI presented to the user (Fig. 5b). Since the UI partials are generated from the elements contained in the AO, the user input clearly relates to the corresponding ontology elements. This allows to build *an instance model* for each presented AO containing the input data of the user using an *Ontology Mapper* (Fig. 5, ❸). The result is a set of ontology instances on which a reasoner can build inferences and derive a *complex product request*, which

can be sent to the *Distributed Market Space* for further processing (i.e. generating a quote/proposal for the requested product components).



a) Basic Architecture (modified from [10]) **b)** sample UI for a travel booking request

Figure 5: Generic UIs for complex product requests

Results: Although the demonstrator is yet still a proof-of-concept, it already showed, that it is possible to share application descriptions and thus provide generic UIs based on Application Ontologies. AOs can be assembled from arbitrary sources (e.g. topic-related repositories for *insurance*, *travel planning* etc.). The UIs can automatically be derived and aggregated based on the contained information. As a second result the demonstrator showed, that these ontological descriptions can be used for non-UI-specific purposes like reasoning on instances of the AOs and thus further processing in backend systems that do understand the used ontologies.

6 Related Work

The research on the automatic generation of UIs covers many contributions during the last years that are based on model-driven concepts. There are several approaches focusing on different aspects of UI generation.

User Interface Description Languages (UIDL) focus mainly on the description of concrete UIs in a technology independent way. Examples are *JavaFX*[3], *UIML* [1], *UsiXML* [13] and *XForms*[4]. The essential idea is to model dialogs and forms by using technology independent descriptions of in-/output controls and relations between elements and behavior (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[23]*, *MARIA* [16] and conversation based

[3] I. Fedortsova et al, 2014 : http://docs.oracle.com/javase/8/javafx/fxml-tutorial/preface.html
[4] M. Dubinko et al, 2003: http://www.w3.org/TR/xforms

approaches by Popp et al. e.g [19]. They focus on a concrete model of the dialog flows and their variants. 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* [2] and *Mecano* [20] which use a **domain model** as starting point for the derivation of UIs. While JANUS is designed to provide CRUD-like interfaces 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. [24] propose an ontology driven framework to describe UIs based on concepts stored in a knowledge base. Khushraj et al. [12] 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. [7] use a profiled domain model enriched with UI related data to describe a UI and associate it with an ontology driven task model. *ActiveRaUL* [21] combines an UIDL with a data-centric approach and makes a significant contribution to the generation of UIs based on arbitrary ontologies. They derive a hierarchical presentation of an ontology and map it to an ontological UI description. Since there is not much semantic information contained, the resulting UIs are yet very simple and not very feature rich regarding the supported interactions.

Dissociation: A main goal of the proposed data-centric approach is to minimize the number of needed artefacts and to use a representation that can be reused for different purposes. 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 modeled 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. That impedes their reuse for other purposes related to application generation.

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. Though this approach is restricted to interview applications, it allows a significantly simplified modelling process, since the results can be derived from a single source.

7 Conclusion

In this paper a *data-centric, model driven approach for the automatic generation of user interfaces* for dialog based *Interview Applications* is presented. The approach is based on a UI-agnostic, data-centric description for applications. The foundation is a model of the processed application data which is enhanced by type-related, structural and behavioral information to yield automatically generated UI variants as demonstrated in the previous sections. The information needs are identified and a meta-

model is derived from which the UIs can be inferred. Furthermore, the information needs are mapped to an ontological description relying on RDF/OWL constructs to get a non-proprietary representation of that information to be used in different contexts. A process to derive UIs from such a data-centric model is outlined. Finally, the evaluation is presented which (1) provides an implementation of the generation process for UIs from data-centric application descriptions, is used as proof-of-concept regarding the (2) usefulness of the ontological descriptions for UI generation and (3) its viability as sharable, non-proprietary means for generating UIs for data-driven applications.

The results of the evaluation indicate, that using a data-centric model is feasible for UI generation in case of Interview Applications. Since the number of artefacts is reduced to a single, UI-agnostic application model, the step for declaring UIs manually can be eliminated. Because of its data-centric nature, it can be used for non-UI-specific tasks. Using a universal representation as RDF/OWL adds even more value, as the application model is sharable and the contained semantics can be exploited by standard tools for reasoning on the model and instances.

The approach is intentionally restricted to dialog based Interview Applications that are very important and frequently used in EIS, e.g. in the insurance domain. Since a limited set of applications was used for the analysis, we cannot claim completeness of the identified interaction patterns. The practical use of the approach will 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 is restricted to hierarchical structures and uses proprietary annotations and thus restricting its universality. Future work might concentrate on finding more general ways for incorporating the information and exploit existing approaches to apply the approach to arbitrary ontologies.

References

[1] Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., Shuster, J.E.: UIML: An appliance-independent XML user interface language. In: WWW '99 Proceedings of the eighth international conference on World Wide Web. pp. 1695–1708 (1999).

[2] Balzert, H., Hofmann, F., Kruschinski, V.: The JANUS Application Development Environment—Generating More than the User Interface. In: Computer Aided Design of User Interfaces, Vol. 96. pp. 183–206 (1996).

[3] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: The CAMELEON Reference Framework. (2002).

[4] Chlebek, P.: User Interface-orientierte Softwarearchitektur. Vieweg & Sohn Verlag, Wiesbaden (2006).

[5] Constantine, L.L., Lockwood, L.A.D.: Software for use: a practical guide to the models and methods of usage-centered design. ACM Press/Addison-Wesley Publishing Co., New York (1999).

[6] Coutaz, J.: 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 (2010).

[7] Gaulke, W., Ziegler, J.: Using profiled ontologies to leverage model driven user interface generation. Proc. 7th ACM SIGCHI Symp. Eng. Interact. Comput. Syst. - EICS ’15. 254–259 (2015).

[8] Hitz, M.: mimesis : Ein datenzentrierter Ansatz zur Modellierung von Varianten für Interview-Anwendungen. In: Nissen, V., Stelzer, D., Straßburger, S., and Fischer, D. (eds.) Proceedings - Multikonferenz Wirtschaftsinformatik (MKWI) 2016. pp. 1155–1165 (2016).

[9] Hitz, M.: Interner Projektbericht zu mimesis.ui. , DHBW-Stuttgart (2013).

[10] Hitz, M., Radonjic-Simic, M., Reichwald, J., Pfisterer, D.: Generic UIs for requesting complex products within Distributed Market Spaces in the Internet of Everything. In: Buccafurri, F. (ed.) CD-ARES 2016, LNCS vol. 9817. (in preparation) Springer (2016).

[11] Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S.: OWL 2 Web Ontology Language Primer, http://www.w3.org/TR/2009/REC-owl2-primer-20091027/.

[12] Khushraj, D., Lassila, O.: Ontological approach to generating personalized user interfaces for web services. Semant. Web–ISWC 2005. 916–927 (2005).

[13] Limbourg, Q.: USIXML: A User Interface Description Language Supporting Multiple Levels of Independence. In: Matera, M. and Comai, S. (eds.) ICWE Workshops. pp. 325–338. Rinton Press (2004).

[14] Meixner, G., Paternò, F., Vanderdonckt, J.: Past, Present, and Future of Model-Based User Interface Development. i-com. 2–11 (2011).

[15] Miguel, A., Faria, J.P.: Automatic Generation of User Interface Models and Prototypes from Domain and Use Case Models. In: Rita Matrai (ed.) User Interfaces. InTech (2010).

[16] Paterno, F., Santoro, C., Spano, L.D.: Maria: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environment. ACM Trans. Comput. Interact. 16, (2009).

[17] Peffers, K., Rothenberger, M., Tuunanen, T., Vaezi, R.: Design Science Research Evaluation. Des. Sci. Res. Inf. Syst. Adv. Theory Pract. 398–410 (2012).

[18] Peffers, K., Tuunanen, T., Gengler, C.E., Rossi, M., Hui, W., Virtanen, V., Bragge, J.: The Design Science Research Process: A Model for Producing and Presenting Information Systems Research. Proc. Des. Res. Inf. Syst. Technol. DESRIST’06. 24, 83–106 (2006).

[19] Popp, R., Falb, J., Arnautovic, E., Kaindl, H., Kavaldjian, S., Ertl, D., Horacek, H., Bogdan, C.: 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 (2009).

[20] Puerta, A.R., Eriksson, H., Gennari, J.H., Musen, M.A.: Beyond data models for automated user interface generation. In: In Proceedings British HCI’94 (1994).

[21] Sahar, A., Armin, B., Shepherd, H., Lexing, L.: ActiveRaUL : Automatically generated Web Interfaces for creating RDF data. 0, 100 (2013).

[22] Sein, M.K., Henfridsson, O., Rossi, M.: Action Design Research. MIS Q. 35, 1–20 (2011).

[23] Van den Bergh, J., Luyten, K., Coninx, K.: 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 (2011).

[24] Liu, B., Chen, H., He, W.: Deriving user interface from ontologies: A model-based approach. Proc. - Int. Conf. Tools with Artif. Intell. ICTAI. 2005, 254–259 (2005).