Generic UIs for requesting complex products within Distributed Market Spaces in the Internet of Everything

Michael Hitz[1], Mirjana Radonjic-Simic[2], Julian Reichwald[2], and Dennis Pfisterer[3]

[1] Baden-Wuerttemberg Cooperative State University Stuttgart, Germany
michael.hitz@dhbw-stuttgart.de,
[2] Baden-Wuerttemberg Cooperative State University Mannheim, Germany
{mirjana.radonjic-simic, julian.reichwald}@dhbw-mannheim.de,
[3] Institute of Telematics, University of Lübeck, Germany
pfisterer@itm.uni-luebeck.de

Abstract *Distributed Market Spaces (DMS), refer to an exchange environment in emerging Internet of Everything, that supports users in making transactions of complex products; a novel type of products made up of different products and/or services that can be customized to better fit the individual context of the user. In order to express their demand for a particular complex product in a way that is interpretable by the DMS, users need flexible User Interfaces (UIs) that allow context-focused data collection related to the complexity of the user's demand. This paper proposes a concept for generic UIs that enables users to compose their own UIs for requesting complex products, by combining existing UI descriptions for different parts of the particular complex product, as well as to share and improve UI descriptions among other users within the markets.*

Keywords: Automatic User Interface Generation, Semantic Web, Internet of Everything, User Interface Ontologies, Commercial Exchange, Distributed Market Spaces, Complex Products

1 Introduction

Emerging Internet of Everything (IoE) is opening up new opportunities for commercial exchange, giving the rise to novel types of products and services. Due to the increased interconnectivity of its participants (companies, institutions, individuals) on one hand and processes, data and things on the other [3], the IoE is enabling exchange environments, where products and services are customized and compound, as they are made up of many components provided by different suppliers [6]. Furthermore, these products and services can be orchestrated in complex products (i.e., an arbitrary combinations of individual products and/or services) and customized in a way to consider the unique conditions determined by the user's context. As such, complex products can better fit the individual

needs of the users, thus, create richer consumer experiences that have not been possible before.

Contemporary solutions for commercial exchange are mostly focused on availability of individual products and services within their domain boundaries, or certain pre-defined combination of them traditionally bought together, however, are limited in their ability to support complex products, which need to fulfill particular user-defined criteria, going beyond the existing product/service descriptions. Consider the simple use case of booking a flight, hotel, rental car and guided tour. While already feasible today, it is a complex task to solve in order to fulfill different constraints (e.g., place, time, price, personal preferences). It can get exceptionally complex if many auxiliary conditions or products are involved. To make informed decisions, users need to know where and how to find viable product and/or service offers i.e., to engage search engines, visit diverse online platforms, shops, etc. while confronting with plenty of different user interfaces, search/selection criteria and representations of product/service description. After finding viable offers, users must compare, aggregate and infer all relevant information, considering the particular user context. The complexity of above mentioned activities and related user involvement lead to the adverse selection [2] i.e., choosing good enough, instead of optimal products/services, and increases the transaction costs (i.e., the buyers' costs to acquire information about seller prices and product offerings).

Distributed Market Spaces (DMS) proposed by [18], refers to a IoE exchange environment that supports market participants (i.e., consumers and producers) in making (distributed) transactions for complex products. But, to build a complex product requests on their own, the users need an alternative *to express their demand for a particular complex product, in a way, that is interpretable by the DMS*. Therefore, flexible user interfaces are needed to allow data collection of an arbitrary combination of the products and/or services, fulfilling user-defined criteria and spanning over different product/service domains related to the complexity of the user's demand.

In this paper, we propose a concept for *generic user interfaces (UIs) for requesting complex products within Distributed Market Spaces in the Internet of Everything* – a concept that alleviates the effects of adverse selection by supporting the users crafting complex product requests in a seamless manner; a manner of enabling users to:

- compose a new, customized UI for requesting a complex product in a particular user-defined context, by combining existing UI descriptions for different parts of the complex product, which can be rendered for different platforms / technical contexts (e.g., mobile or webbased apps)
- create a request for complex products interpretable by the DMS and
- share and improve UIs for complex products within markets.

This paper is organized as follows: First, Section 2 describes the setting in which our proposed concept is applied and defines the main requirements. Next, Section 3 presents the architecture and functional structure of the proposed solution, followed by a demonstrator implementation in Section 4. Thereafter, Section 5

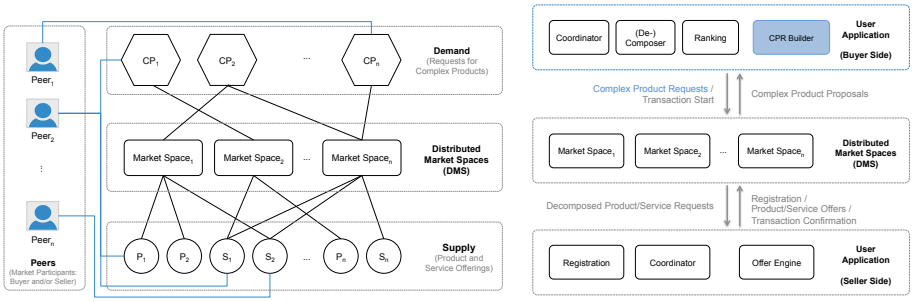
discusses on related work and Section 6 concludes the paper with a summary and outlook.

2 Motivation and Background

In the following, we briefly describe the setting, i.e., the context in which the proposed concept of generic UIs is applied. Afterwards, we define the overall objectives and consider these as the requirements for the demonstrator implementation, as shown later in Section 4.

2.1 The Application Context

Distributed Market Spaces (DMS) [18], refers to a model of commercial exchange that supports market participants in making distributed transactions of complex products. Figure 1 illustrates the conceptual structure of the DMS, showing the involved parties, their roles and relationships on the left, and on the right, the DMS functional structure with its components and high-level interfaces, represented through the sets of exchanging messages, required to support the interactions along involved parties.



(a) Market-level architecture (b) Functional structure

Figure 1: Conceptual structure of the DMS.

As shown in Figure 1a, the DMS is used by peers, i.e., potential transaction partners defined by their intention; buyers are peers intending to buy complex products while sellers are peers intending to sell products and/or services. Peers connect to one or more independent market spaces (MS) and multiple of these market spaces form the Distributed Market Spaces (DMS). A peer may offer products and services on one or more market spaces (e.g., $Peer_2$ offers product P_1 and service S_2) as well as, request a complex product by sending the request to one or more market spaces (e.g., $Peer_2$ also requests the complex product CP_1).

The peer interface component (user application), as shown in Figure 1b, is split into two parts: a seller and a buyer side. In this paper, we focus on the user

application *buyer side*, which is responsible to: transform a user's intention into a complex product request, distribute these requests to multiple market spaces, receive (partial) complex product offers and re-combine them into multiple complete complex product proposals, rank them according to the buyer's context and requirements, as well as to coordinate the distributed buying transaction. The user application is therefore comprised of the CPR Builder, Coordinator, the (De-) Composer and Ranking component.

The proposed concept of generic UIs is applied in the context of Complex Product Request Builder, CPR Builder component highlighted blue (Figure 1b).

2.2 Overall Objectives

The main task of the CPR Builder as a component of the user application, is to transform a user's intention into a complex product request, which can be distributed to one/multiple market spaces within the DMS.

For the generic UIs concept to be implemented in context of the CPR Builder, following two basic prerequisites need to be assumed. First, in order to be able to match the incoming requests with the available offerings, the market space needs to understand the semantics of the product/service offerings (i.e., 'supply semantics'). Second, it needs to understand the semantics of the different requests, provided by the user as a complex product request (i.e., 'demand semantics').

As to [18], the DMS supports the 'supply semantics' using a domain-agnostic database, containing the information about registered sellers, as well as the description of available products and services they can potentially offer, encoded in RDF [24].

The challenge at this point, is the possible gap between the product/service descriptions provided by the supply-side and the demand descriptions requested by the demand-side.That is, because the data to describe a complex request is usually different from the data contained in the product/service descriptions. For example, a product description usually contains a *price tag* for one unit – a demand usually contains a *price range*. A more complex example is the generalisation of the demand: a product description could be a specific offering for a Ticket for the musical 'Chicago' – a general demand for a concert could be 'all musicals and rock concerts' that somehow are related to 'Chicago'. Given that, it is not always possible to use the product/service description data as a blueprint for the demand requests. Therefore, the DMS uses dedicated demand descriptions, defining the demand in a way, that can be mapped to the descriptions of the offerings.

As the main purpose of the CPR Builder is to enable users to craft complex product requests in a seamless manner, the potential buyers should be able to combine different product/service requests into a single complex product request, hence to compose a specific UI variant that:

- is tailored to their demand for a particular complex product – i.e., an arbitrary combination of products and/or services, e.g., a flight, hotel, rental car and tickets for the events at the destination,

- supports the context-information defined by the user, i.e., user's preferences and criteria e.g., a ticket for a certain musical vs. more general proposal for events like theater, concert or ship cruise, and
- produces output in the form of a request for complex product interpretable by the DMS.

Composing a specific UI variant includes finding, selecting and combining existing UI descriptions for the different parts of the complex products, while considering the user's preferences in terms of the generalisation (e.g., concert vs. general event planning) or granularity (e.g., less questions vs. detailed specification depending on the user's context). Hence, we can detail this overall objective into following functional requirements:

R_1**:** *Different UI descriptions for the same demand request* supplying different questions based on the user's preference for more/less specific questions.
R_2**:** *Different UI descriptions containing demands for multiple products* to allow context-focussed interfaces.
R_3**:** *Composition of different UI descriptions into a single UI description* for the particular complex product request

Having outlined the main prerequisites and requirements, in following, we use them as the rationale for the conceptualisation of the overall solution.

3 Proposed Solution

As an explanation of the proposed solution, in this section, we first introduce the foundations and core elements, followed by the functional aspects and more detailed description of the inner workings.

3.1 Overview - Generic UIs for complex product requests

The proposed solution extends the DMS concept, outlined in Section 2.1, by enabling users to craft complex product requests. It uses the Complex Product Builder to combine individual UIs and generate complex product requests from the collected data. For the automatic generation of the involved UIs, the solution builds on the results of the mimesis project [10] and extensions that map the approach to ontological descriptions [11].

In the proposed solution of generic UIs, we use Semantic Web technologies as they provide the necessary mechanisms to get a rich description of the data involved and incorporate techniques for reasoning on that data. The foundation of the proposed solution is built on ontologies describing different views of the participants (demand-, DMS- and supply-side) and ontological descriptions for the UIs to meet the requirements R_1,R_2 and R_3 (cf. Section 2.2).

Figure 2 shows the core elements of the proposed solution. The central component is the **Complex Product Builder**, that orchestrates the generation of the UI. It allows the selection of UI descriptions, the generation of their concrete

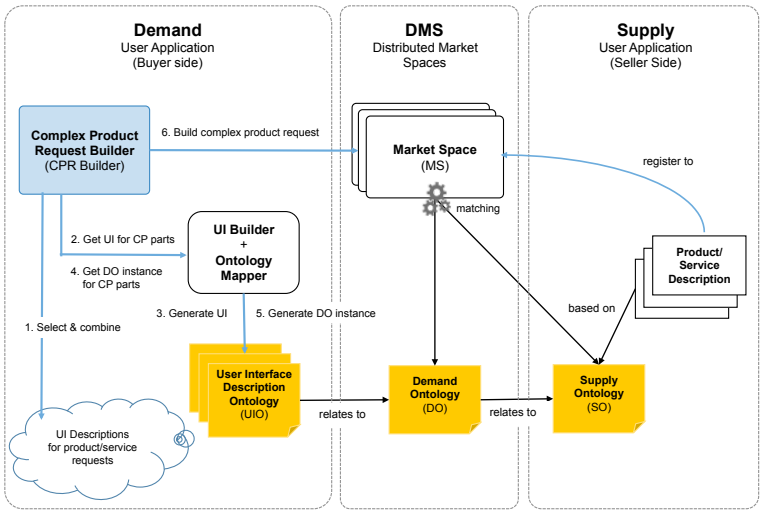


Figure 2: Core elements of the solution architecture.

UIs, aggregation into a single UI (similar to 'mesh-up' approaches [16], though using generated content) and building of a complex product request based on the data entered by the user.

A **Supply Ontology (SO)** is used by the peers (seller side) to describe product instances on which market spaces within the DMS can operate. To describe the demand for a product/service, a **Demand Ontology (DO)** is used; it defines the possible request data to be specified by the peers (buyer side) and thus, can be interpreted by the market spaces. Finally, **User Interface Description Ontologies (UIO)** describe the UI variants based on the data to be collected. These are used to build the concrete UI and to generate output, that corresponds to the related demand ontologies.

The following Section 3.2, outlines the functional structure of the solution needed to fulfil defined requirements (R_1, R_2, R_3), followed by Section 3.3, that provides a detailed description of the ontologies, used in this solution – especially focussing the UIO and its mapping to the DO.

3.2 Functional View - Processing complex product demands

The workflow for building a complex product request starts with the buyer aggregating the UI for a specific complex product need. As shown in Figure 2 (Step 1), the buyer selects suitable, task related UI descriptions provided by a **UI Description Repository** – e.g., a search engine collecting UI descriptions for demands on the Internet, or a repository of community-rated UI descriptions (which usually were manually or semi-automatically crafted based on the related DOs). The result of this step, is a collection of user-selected, context-related UI descriptions to be presented to the user by the CPR Builder.

The collection of UI descriptions is sent to the **generic UI Builder** component, that generates the final UIs based on that descriptions and returns the results. The components are aggregated by the CPR Builder into a single UI and are presented to the user (Steps 2 and 3). When the user finished entering data, the data for each UI component is mapped to instances of the corresponding DO (Steps 4 and 5). The information on how the data elements relate to DO elements is part of the UIO (cf. Section 3.3). The resulting DO instances are aggregated into one complex product request, that is enriched with context data and thus, as shown in Step 6, ready for further processing.

3.3 Information View - Ontologies in detail

Supply Ontology (SO): The Supply Ontology is the common vocabulary to describe products/services provided by a seller. As in [18], this enables a market space to process product/service data and clearly determines the data and semantics that can be used for a certain product/service instance description. A simple product instance is shown in Listing 1.1 as an example. It describes a ticket offer for the musical 'Chicago' at the 'Alte Oper Frankfurt' [18] using existing ontologies *GoodRelations*[4] and *Ticket* ontology[5].

Listing 1.1: Exemplary description of a offering.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gr: <http://purl.org/goodrelations/v1#>
PREFIX tio: <http://purl.org/tio/ns#>
PREFIX dms: <http://www.itm.uni-luebeck.de/dms/#>

dms:ticket1 a tio:TicketPlaceholder ;
  rdfs:label "Ticket␣for␣Chicago␣Musical␣at␣Alte␣Oper␣Frankfurt"@en ;
  tio:accessTo <http://data.linkedevents.org/event/chicagomusical> .
  dms:TRIO Tickets ltd. gr:offers dms:PSO1 .
  dms:PSO1 a gr:Offering ;
  gr:name "Ticket␣for␣Chicago␣Musical"@en ;
  gr:description "The␣#1␣American␣Musical␣in␣Broadway␣History:Chicago␣at␣Alte␣Oper␣
       Frankfurt"@en ;
  gr:includes dms:ticket1 ;
  gr:hasBusinessFunction gr:Sell ;
  gr:hasPriceSpecification
    [ a gr:UnitPriceSpecification ;
     gr:hasCurrency "USD"@en ;
     gr:hasCurrencyValue "49.50"^^xsd:float ;
    gr:validThrough "2016-09-26T23:59:59"^^xsd:dateTime ] .
```

Demand Ontology (DO): The Demand Ontology is closely related to the Supply Ontology and defines the data that can be used to describe the demand for a certain product (e.g., concert ticket) or product category (e.g., ticketing in general). The DO describes the maximum of requestable information, thus, what the market space is *able to understand and handle* regarding the questions

[4] http://www.heppnetz.de/projects/goodrelations/
[5] http://www.heppnetz.de/ontologies/tio/ns

Listing 1.2: Example of a request for a ticket.

```
@prefix : <http://mimesis.solutions/products/concert/individuals#> .
@prefix ... owl:  rdf:  xml:  xsd:  rdfs:
@prefix gr: <http://purl.org/goodrelations/v1#>
@prefix tio: <http://purl.org/tio/ns#>
@prefix tido: <http://demandontologies.org/ticketdemands#>
@base <http://mimesis.solutions/products/concert/individuals> .

###  http://mimesis.solutions/products/concert/individuals#_i1462530726859
:_i1462530726859 rdf:type owl:NamedIndividual ;
  :concertdata :ticketrequest_i1462530726859 .

###  http://mimesis.solutions/products/concert/individuals#ticketrequest_i1462530726859
:ticketrequest_i1462530726859 rdf:type <tido:TicketRequest> ,
  owl:NamedIndividual ;
  gr:name "Chicago" ;
  tido:eventcategory "musical|rockconcert"^^<tido:eventcategorylist> ;
  gr:hasPriceSpecification :hasPriceSpecification_i1462530726859 .

###  http://mimesis.solutions/products/concert/individuals#
     hasPriceSpecification_i1462530726859
:hasPriceSpecification_i1462530726859 rdf:type <gr:UnitPriceSpecification> ,
  owl:NamedIndividual ;
  gr:hasMaxCurrencyValue "35"^^<xsd:float> .
```

related to the corresponding Supply Ontology. Listing 1.2 shows an example of a request following a DO for event tickets. It describes a demand for a ticket for a *musical or rock concert* with a name containing 'Chicago' (could be the musical or the band).

Although, the DO is related to a corresponding SO (here the Ticket Ontology), it extends the elements with request-related extensions (e.g., *tiod:eventcategory*; for a more general demand for event categories, or the use of *gr:hasMaxCurrency-Value* for specifying a maximum price for a ticket.

User Interface Description Ontologies (UIO): A set of User Interface Description Ontologies is used to describe the possible UIs for the buyer side. A UIO describes the UI for a **specific dialog variant** by describing the **data to be collected** in sufficient detail. It contains all necessary information needed to (1) derive a User Interface and (2) to relate the collected data to a demand instance specified by Demand Ontologies.

For **the description of UIs** we apply the approach of the mimesis project introduced in [10] and its application onto ontologies [11]. The basic idea of the approach is to define a model of the data processed/collected by the application and derive UIs for different platforms and user contexts from this model. For this purpose, the basic data model is enriched with information needed to derive UIs: this includes *structural information* (e.g., type restrictions, grouping, sequence of elements) and *behavioural information* (e.g., visibility rules, reactions to the changes, validations to perform). The resulting model is data centric, technology agnostic and can be used to derive UIs for different kinds of platforms and contexts of use (e.g., mobile apps, web based-, rich client- or speech based UIs). Further details of the approach can be found in [10].

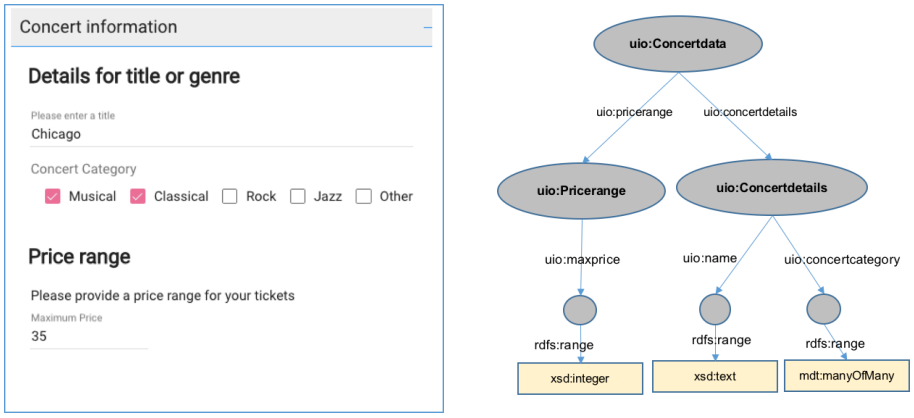
The idea of the mimesis approach – to describe the semantics of the data in more detail – additionally allows to add information for each element on

how it is to be mapped to elements defined within a DO. Using that information an instance (individual) of the UIO containing the user input can be used to generate a corresponding instance of the DO as resulting output.

The approach is suitable to meet the requirements R_1, R_2 and R_3 listed in Section 2.2. To achieve that, ontological descriptions of the UI for different product demands are used. These contain:

- a description of the **data with UI specific enhancements**, as defined in mimesis for the derivation of UIs (as proposed in [11]) and
- the information for the **mapping of that data onto DO instances,** needed to produce the demand requests.

Hereby, different UI variants for a demand can be defined, that might contain different questions depending on the user's context (cf. R_1). Since the mapping information contained in the UI description can reference arbitrary ontologies, it is also possible to provide UIs containing questions spanning different DOs (cf. R_3). The approach is also capable to address requirement R_3: the CPR Builder is able to choose a set of different UIOs as building blocks from which an aggregated UI can be presented to the user.



(a) UI Variant for a concert demand.

(b) Graph of the UIO.

Figure 3: UI and its structure.

Figure 3a shows a possible variant for a UI relating to the above example DO instance, in Listing 1.2. The UI for a *concert demand* contains two groups of questions (*Details for title or genre* and *Price range*). It illustrates, that the relation to the DO is not one-to-one. For example, the data are grouped differently and there is no currency selectable (which is already set by the CPR Builder from the user's context data). Additionally, there is a value restriction for *concert category*, which might be a subset of the possible values defined in the DO.

Figure 3b shows a structural graph of the UIO for the displayed UI. The UI consists of two groups (*Pricerange* and *Concertdetails*). These encompass the

Listing 1.3: Basic User Interface Ontology for a concert demand (excerpt).

```
@prefix : <http://mimesis.solutions/products/concert#> .
@prefix owl: rdf: xsd: rdfs:
@prefix mdt: <http://mimesis.solutions/datatypes#>.
@prefix man: <http://mimesis.solutions/annotations#>
@base <http://mimesis.solutions/products/concert> .
<http://mimesis.solutions/products/concert> rdf:type owl:Ontology .

###########    Classes     ##########
:Concertdata rdf:type owl:Class .
:Concertdetails rdf:type owl:Class .
:Pricerange rdf:type owl:Class .
...
###########    Object Properties     ##########
:Concertdata.concertdetails rdf:type owl:FunctionalProperty , owl:ObjectProperty ;
  rdfs:domain :Concertdata ; rdfs:range :Concertdetails .
:Concertdata.pricerange rdf:type owl:FunctionalProperty ,owl:ObjectProperty ;
  rdfs:domain :Concertdata ; rdfs:range :Pricerange .
...
###########    Data properties    ##########
:Concertdetails.concertcategory rdf:type owl:DatatypeProperty , ... ;
  rdfs:range  mdt:manyOfMany ; rdfs:domain :Concertdetails .
:Concertdetails.name rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
  rdfs:range xsd:text ; rdfs:domain :Concertdetails .
:Pricerange.maxprice rdf:type owl:DatatypeProperty , owl:FunctionalProperty ;
  rdfs:domain :Pricerange ;  rdfs:range xsd:integer .
    ...
```

data fields and their types (e.g., *name* and *concertcategory*) to be presented to the user. Listing 1.3 shows an excerpt of the UIO in OWL/Turtle notation [23].

The additional information needed for the derivation of a concrete UI and for the mapping to the Demand Ontology is shown in Listing 1.4. As this is meta information, describing the element in more detail, mimesis uses the annotation concept of OWL to specify these details. For each element (data element or group) there exist mimesis-specific entries (e.g., the *sequence* of the questions in line 3, or specific *type information and restrictions* in line 4 and 5). The mapping onto instances for DO elements is provided using annotations starting with the prefix '*sw:*'. It contains information to which class an entity belongs to (e.g., line 19 maps *Concert.concertdata* to a *tiod:TicketRequest*). It is defined to which property a data element maps, which type it has and to which individual it belongs to (e.g., line 6 - 8 map *Concertdetails.concertcategory* to the type *tiod:eventcategorylist* and assigns it to the *ticketrequest* instance using the property name *tiod:eventcategory*). Given that information a demand instance, following the DO as shown in Listing 1.2, can be generated in combination with the instance data gathered by the UI.

4 Demonstrator

As a proof of the concept, we built a demonstrator that implements the proposed approach for aggregating the UIs for a complex product request and matching the collected data to a Demand Ontology. As a use case, we chose 'organising a city trip' which includes the planning of events, transportation and overnight stays. The buyer should be able to select the desired components for his trip, and enter the required demand information for each component. As the final

Listing 1.4: Additional UI and DO related Data for a concert demand.

```
###########    Annotations    ###########
:Concertdetails.concertcategory man:sequence "2" ;
  man:type "manyOfMany" ;
  man:restrictedTo>"musical|classical|rock|jazz|all" ;
  man:swForIndividual "ticketrequest" ;
  man:swProperty "tiod:eventcategory" ;
  man:swType "tiod:eventcategorylist" .
:Concertdetails.name man:sequence "1" ;
  man:swProperty "gr:name" ;
  man:swType "gr:name" ;
  man:swForIndividual "ticketrequest" .
:Concertdata.pricerange man:sequence "2" ;
  man:swClass "gr:UnitPriceSpecification" ;
  man:swProperty "gr:hasPriceSpecification" ;
  man:swIndividual "hasPriceSpecification" ;
  man:swForIndividual "ticketrequest" .
:Concert.concertdata man:sequence "0" ;
  man:swClass "TicketRequest" ;
  man:swIndividual "ticketrequest" .
:Pricerange.maxprice man:sequence "1" ;
  man:initialValue "30" ;
  man:unit "EUR" ;
  man:swProperty "gr:hasMaxCurrencyValue" ;
  man:swForIndividual "gr:hasPriceSpecification" ;
  man:type "number" ;
  man:swType "xmls:float" .
:Concertdata.concertdetails man:sequence "1" .
...
```

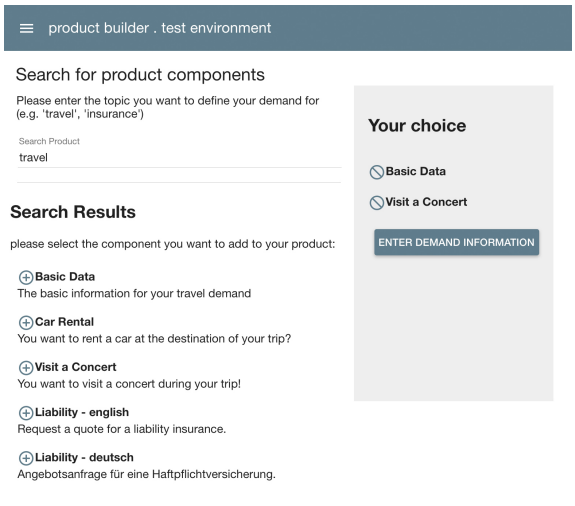
step of the demonstrator, the collected data for each component is shown as an instance of the related Demand Ontologies.

The demonstrator implements the solution architecture outlined in Section 3.2 and shown in Figure 2. Its central component, the CPR Builder, is implemented as a web application using HTML/JavaScript as a platform technology, and uses a local repository for the management of available UI descriptions. Additionally, the *UI Builder and Ontology Mapper* components are implemented as separate Web Services, based on the work in [11].

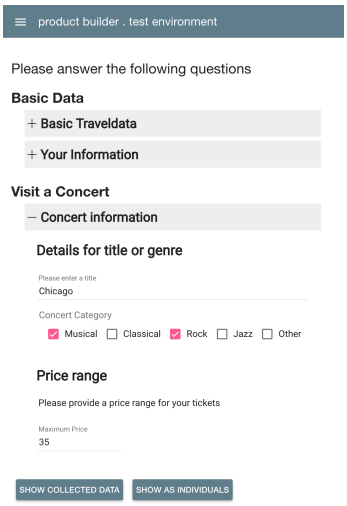
The UI Builder Service is responsible for generating the UIs based on the UI Description Ontologies; it accepts UIOs as input and is able to generate a final UI for different technology plattforms (here HTML/Javascript). The Ontology Mapper Service, on the other hand, is responsible for generating a DO instance, based on a certain UIO and corresponding user input submitted in the form of a JSON object.

The demonstrator follows the workflow outlined in section 3.2. First, the buyer searches for product components using a google-like search facility, and selects/collects the components according to his preferences and requirements as shown in Figure 2, Step 1. An example of such a search is shown in Figure 4a, where the user enters 'travel' into the search field, and gets available components matching the search critera (e.g., 'visit a concert' of 'rent a car'). The user selects viable components, which are collected like products in a *shopping cart* (Figure 4a). When the user finished the selection, the corresponding UIOs of the selected components are sent to the **UI Builder Service** that generates final UIs (Figure 2, Steps 2 and 3). These are aggregated by the client application into one UI and

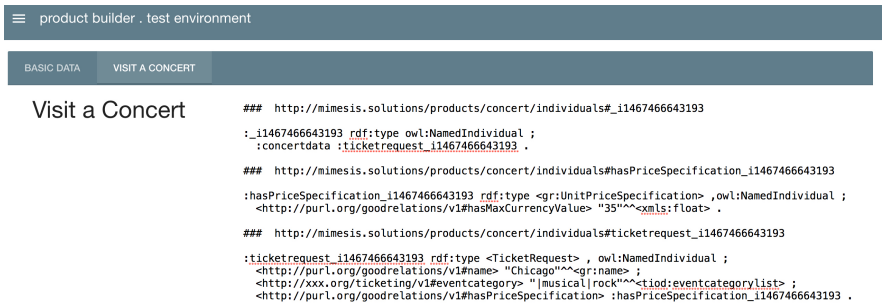
presented to the user for input. The aggregation of the UI based on the selected components is shown in Figure 4b.



(a) Searching for suitable UIs.



(b) Entering data in meshed-up UI.



(c) Gathered data as Demand Ontology instance.

Figure 4: Aggregating UI and entering data

After having entered the demand data for each component, the CPR Builder sends the collected data along with the corresponding UIO to the **Ontology Mapper Service**, which is responsible for mapping the collected data to the DO according to the information contained in the UIO, and returns an instance of the DO (Figure 2, Steps 4 and 5). The results of this step are finally displayed by the demonstrator for each component as shown in Figure 4c. In a further step (not part of the demonstrator) these DO instances can be aggregated into one complex demand and processed by the CPR Builder, as described in Section 2.1.

Summarizing, the demonstrator shows that it is basically possible to dynamically generate UIs for complex products, satisfying the requirements defined in

Section 2.2. Users can be enabled to choose the UIs they want/need and combine them to build requests for desired complex products. These UIs may span different domains and can be combined to build a unified interface for the users. Moreover, the demonstrator also underlines the advantage of using UIOs as separate descriptions; since a UIO contains all information to generate a UI, as well as the information about how to produce instance data (understandable by the market spaces), the UIOs are actually independent from the target system. Therefore, they can be independently distributed and modified – as long as the output conforms to the specified demand ontologies.

Yet, our demonstrator does not cover all functionality needed to combine different demands in a seamless manner; currently it combines UIs for product components as separate, self relying units – ignoring possible relations between them. For example, it does not implement a context, which components and their UIs may share and react to (e.g., recognizing and omitting questions, that were already asked in other components, or pre-fill values from a global context).

5 Related Work

In this section, we provide an overview of the operational solutions, concepts and approaches relevant to the presented work, and briefly discuss why these are not suitable to meet the defined requirements.

Electronic marketplaces (e-marketplaces), as well-established solutions for commercial exchange, enable only compositions of individual products/services within their domain boundaries, or they offer pre-defined combinations of them, which are traditionally bought together and determined by recommender systems. Even though there exist some advanced solutions, such as, e.g., [8] enabling the composition of individual services considering a wider set of user-defined criteria, these are domain-specific solutions, and as such, are limited in their capabilities to support users requesting complex products spanning over different product/service domains related to the complexity of the user's demand.

The Intention Economy (IE) [22], also called Project Vendor Relationship Management (VRM), refers to an exchange environment that focuses on a buyers' intention to conduct a transaction with potential sellers (i.e., vendors). By using VRM tools, buyers are supported to describe their needs by creating a personal request for proposal (pRFP) and make them visible for the vendors. Even though, the VRM tools support pRFP there is no obvious evidence that they support composing context-focussed UIs.

Web of Needs (WoN) [14], refers to a framework for a distributed and decentralized e-marketplace on top of the Web. WoN aims to standardize the creation of owner proxies, which describe supply or demand, represent the intention to enter a transaction, as well as contain information of the owner needed for conducting the transaction [13]. Generally, WoN supports describing the user's need for complex products, but, if the user wants the system to process the complex product, he can publish a 'complex need', waiting for a matching service

capable of interpreting his 'complex need'. Given that, the effects of adverse selection are still retained, and native support for requesting complex products remains insufficient.

Concluding, contemporary solutions of commercial exchange are limited in supporting users requesting complex products; approaches such as IE or WoN, address some of the requirements, but do not represent a comprehensive solution. Either they provide tools that need to be integrated with other solutions to be fully usable, or they address our requirements only partly.

Next, we elaborate on the approaches focusing on automatic generation of UIs and the different aspects of the UI generation that can be applied to our presented work.

User Interface Description Languages (UIDL) focus mainly on the description of concrete UIs in a technology independent way. Examples are JavaFX [7], UIML [1], and XForms [5]. The essential idea is to model dialogs and forms by declarative 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., MARIA [17] or model conversations, like in [19], [20]. They focus on a 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.

Existing **ontology based approaches** generally rely on the concepts of the mentioned approaches and use ontologies to represent the information. For instance, in analogy of UIDL approaches, Liu et al. [15] propose an ontology driven framework to describe UIs based on concepts stored in a knowledge base. Khushraj et al. [12] use web service descriptions to derive UI descriptions based on a UI ontology, adding UI related information to the concept descriptions. In analogy with task based approaches, Gaulke et al. [9] use a profiled domain model enriched with UI related data to describe a UI and associate it with an ontology driven task model which models the interaction. ActiveRaUL [21] combines an UIDL with a data-centric approach and thus contributes to the generation of UIs for arbitrary ontologies. They derive a hierarchical presentation of an ontology and map it to an – yet simple – ontology based on the UI description.

In view of our requirements (cf. Section 2.2) it can be stated that the aforementioned approaches are restricted mainly to the definition of UIs and dialog flows. They do not contain concepts to associate (map) the collected data to results of arbitrary ontology instances that might have a different structure as in the UI. Additionally, they are restricted to the environments where a reasoner is available at runtime to infer the dynamic behavior of UIs based on already entered data (e.g., showing/hiding UI parts, as in [15]). This is a drawback for environments like web-based, single-page applications, where a reasoner is not available at runtime. Finally, UIs and task models are mostly modeled using a large amount of artifacts, thus, they can hardly be used to generate target-system independent variants that differ in content, depending on the context of use [4].

6 Conclusion and Future Work

In this paper, we proposed a concept for generic UIs that enables users requesting complex products within DMS in the IoE. In order to express their demand for a particular complex product, in a way, that is interpretable by the DMS, users need flexible UIs that allow context-focused data collection related to the complexity of the user's demand.

In order to identify the overall objectives, which need to be supported by the generic UIs concept, we first looked at the prerequisites and objectives derived from the DMS as the application context. Afterwards, we operationalized these objectives into the requirements and used them as the rationale to conceptualize the overall solution as well as to elaborate on existing approaches and initiatives related to the presented work. Thereafter, and in the view of these requirements, we implemented an initial demonstration of the proposed generic UIs concept, using an exemplary use case.

As the demonstrator shows, it is basically possible to dynamically generate UIs for complex products where UIs may span different domains, and can be combined to build a unified interface for the users. It also underlines the advantage of using User Interface Ontologies as separate descriptions, so that they can be independently shared and modified. However, the presented demonstrator does not cover all functionality needed to combine different demands; currently it composes UIs for product/service components as separate, self-relying parts ignoring possible relations between them. Furthermore, it does not consider a wider user-related context that components and their UIs may share and react to.

In our future work, we will concentrate on these two areas of improvements, as well as, on the extensive prototypical implementation to conduct a sophisticated analysis of the strengths and weaknesses of the proposed concept.

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. Akerlof, G.A.: The market for lemons: Quality uncertainty and the market mechanism. The quarterly journal of economics pp. 488–500 (1970)
3. Cisco: The internet of everything for cities. http://www.cisco.com/web/about/ac79/docs/ps/motm/IoE-Smart-City_PoV.pdf (2013)
4. 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. No. Eics (2010)
5. Dubinko, M., Klotz, L., Merrik, R., Raman, T.: XForms 1.0 W3C Recommendation - http://www.w3.org/TR/xforms (2003)
6. El Sawy, O.A., Pereira, F.: Business modelling in the dynamic digital space: An ecosystem approach. Springer (2013)

7. Fedortsova, I., Brown, G.: JavaFX Mastering FXML, Release 8 (2014), http://docs.oracle.com/javase/8/javafx/fxml-tutorial/preface.htm
8. García-Gómez, S., Jimenez-Ganan, M., Taher, Y., Momm, C., Junker, F., Biro, J., Menychtas, A., Andrikopoulos, V., Strauch, S.: Challenges for the comprehensive management of cloud services in a paas framework. Scalable Computing: Practice and Experience 13(3) (2012)
9. Gaulke, W., Ziegler, J.: 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 (2015)
10. Hitz, M.: mimesis : Ein datenzentrierter Ansatz zur Modellierung von Varianten für Interview-Anwendungen. In: Nissen, V., Stelzer, D., Straßburger, S., Fischer, D. (eds.) Proceedings - Multikonferenz Wirtschaftsinformatik (MKWI) 2016. vol. 4, pp. 1155–1165 (2016)
11. Hitz, M., Kessel, T.: mimesis : A Data-Centric Approach for Generating User Interfaces for Interview Applications Using Ontologies. unpublished – in consideration (2016)
12. Khushraj, D., Lassila, O.: Ontological approach to generating personalized user interfaces for web services. The Semantic Web@ISWC 2005 pp. 916–927 (2005)
13. Kleedorfer, F., Busch, C.M.: Beyond data: Building a web of needs. In: Proceedings of the WWW2013 Workshop on Linked Data on the Web (2013)
14. Kleedorfer, F., Busch, C.M., Pichler, C., Huemer, C.: The case for the web of needs. In: Business Informatics (CBI), 2014 IEEE 16th Conference on. vol. 1, pp. 94–101. IEEE (2014)
15. Liu, B., Chen, H., He, W.: Deriving user interface from ontologies: A model-based approach. Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI 2005, 254–259 (2005)
16. Pascalau, E.: Mashups : Behavior in Context (s). In: Proceedings of 7th Workshop on Knowledge Engineering and Software Engineering (KESE7) (2011)
17. Paterno, F., Santoro, C., Spano, L.D.: Maria: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environment. ACM Transactions on Computer-Human Interaction 16(4) (nov 2009)
18. Pfisterer, D., Radonjic-Simic, M., Reichwald, J.: Business model design and architecture for the internet of everything. Journal of Sensor and Actuator Networks 5(2), 7 (2016)
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. Raneburger, D., Kaindl, H., Popp, R., Šajatovi, V., Armbruster, A.: A Process for Facilitating Interaction Design through Automated GUI Generation. In: SAC '14 Proceedings of the 29th Annual ACM Symposium on Applied Computing. pp. 1324–1330. ACM Press, New York (2014)
21. Sahar, A., Armin, B., Shepherd, H., Lexing, L.: ActiveRaUL : Automatically generated Web Interfaces for creating RDF data 0, 100 (2013)
22. Searls, D.: The intention economy: when customers take charge. Harvard Business Press (2013)
23. W3C: Rdf 1.1 turtle. http://www.w3.org/TR/turtle/ (2014)
24. W3C: Resource description framework (rdf). http://www.w3.org/RDF/ (2015)