Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Development of a UI Submodel for the Industry 4.0 Component

L. Baron [*] **A. Braune** [*]

[*] *Technische Universität Dresden, Institute of Automation, 01062 Dresden, Germany; {lukas.baron;annerose.braune}@tu-dresden.de*

Abstract: A provision of user interfaces (UI) by use of the Industry 4.0 component and its asset administration shell (AAS) requires the development of a new UI submodel. Depending on the intended use case and the planned context of use, the submodel needs to be able to store and characterize multiple UI fragments (Variants of the UI) that shall be included in UI solutions. For that, appropriate properties need to be identified and a submodel structure has to be designed. In order to apply the newly designed UI submodel in a plug and produce scenario, the properties need to be formally specified, thus, being interpretable by automatic tools. This contribution presents a UI submodel, a catalogue of UI fragment properties, and a first case study applying these properties to UI fragments of existing industrial systems.

Keywords: Asset Administration Shell, Industry 4.0, Human Machine Interface, Semantics

1. INTRODUCTION

The *Industry 4.0* (I4.0) component is often considered to be the key element for addressing I4.0 use cases, such as the *Plug-and-Produce* (PnP) scenario[6]. It consists of an *Asset Administration Shell* (AAS) managing *assets* that can be any physical or virtual item of value for an organization[11]. The AAS provides *submodels* of the asset and services for communicating with the asset or the submodels.

In a PnP scenario, a technical plant is subject to structural changes that could not be foreseen during plant design[33]. These can be simple changes, such as the replacement of a failed device, but also complex ones, such as the removal or integration of entire plant components, or even the complete creation of a new plant. The objective of the PnP scenario is for the plant and all its automation components to adapt fully automatically to the new plant configuration in order to minimize downtime or commissioning efforts[33]. In such a scenario, the I4.0 components serve as a source of information for updating the process control system.

A *User Interface* (UI), through which human personnel interacts with the plant or with individual components, is also always part of a technical plant, even in future I4.0 systems[35]. Depending on the size and complexity of the system as well as the scope and complexity of the tasks to be performed using UIs, such UIs comprise complex software and hardware components. UIs reflect the plant by representing all information required for the work task which often includes a suitable representation of individual components, but also entire structures consisting of several components, the technical processes, or the manufactured products. Structurally, they are often closely related to the plant, e. g. by reflecting the plant hierarchy and topology in the navigation structure of a *UI solution*. However, in addition to the plant, other factors, referred to as the *context of use*[39], are influencing the design of the UI. This includes *user properties*, properties of the hardware platform on which a UI solution is executed – also *UI platform* – as well as *environmental properties*[13; 15].

As a result of structural changes to the plant, corresponding changes to work processes and the context of use, changes to the UI solutions are unavoidable. New or adapted work tasks must be supported, changes to the plant structure must be adopted and changed context properties must be applied to adapted UI designs. If the UI solution is able to carry out such changes automatically, it can be considered *PnP-capable*. If new system components are put into operation, within the scope of a PnP scenario, it is indispensable to deliver the necessary UI parts – hereinafter referred to as *UI fragments* – coherently to the components so that the UI fragments can be integrated into the UI solution. Our approach is to use the AAS of such process components to deliver the required UI fragments. For that, it is necessary to develop a new *UI submodel* carrying the UI fragments, since none has been proposed in the relevant literature on the AAS[3; 4; 32] yet.

Using the UI submodel, however, not only exactly one UI fragment per component has to be delivered, but many design *variants*, thus, many UI fragments, due to the high variability of requirements for different UI solutions. To differentiate between multiple UI fragments, they must be characterized in the submodel by expressive and unambiguously interpretable *properties*. However, in a development process for PnP-capable UI solutions, the UI fragments shall be evaluated and selected according to their properties for their suitability for the specific UI solution. In the context of PnP scenarios, such a selection should be made automatically by a UI development tool. Therefore, the goal of our work is the development of a UI submodel for the AAS that supports the provision of multiple UI fragments by providing their implementation and a description using well-defined properties that enable the automatic selection of suitable fragments. It shall be noted, that it is not the intention of our approach to describe or replace the implementation of the fragments, thus, the implementation shall still rely on existing UI technologies.

This contribution is structured as follows: In Sect. 2 below, appropriate properties for the UI fragments are identified according to the current requirements for industrial UIs. In Sect. 3, we present excerpts of the current state of the art. After that, the actual UI submodel will be designed in Sect. 4. Finally, in Sect. 5, we present a first case study of UI fragments that are described by use of the identified properties.

2. REQUIREMENTS AND IDENTIFICATION OF RELEVANT PROPERTIES

2.1 Stakeholder and UI Fragment Property Requirements

Several stakeholders are involved in the development and the subsequent use of certain I4.0 components including their UI fragments, cf. Fig. 1. Their respective tasks with regard to the creation of UI fragments or UI solutions, and the use of the UI solutions create several requirements for the UI submodel and its features. In this contribution, only the selection of existing fragments is being considered.

The *plant operators* and the associated personnel interact with the plant using UI solutions. The UI solution must be suitable for the respective *use-case*, i. e. for the concrete *work tasks* of the individuals using it[39]. Furthermore, UI solutions must be adequate for the *contexts* in which they are used. These two influencing factors must be taken into account in the development of usable user interfaces in order to enable efficient and error-free work[14]. The *individual role* of the user within the organization and the *individual user properties*, such as user knowledge and experience in relation to the task and the technical system, are particularly important. However, when assembling a UI solution from separately developed UI fragments, it is crucial to ensure that the solution is consistent from a design perspective[7; 8]. Thus, both content constraints (object and scope of the UI) and design constraints (e. g. used symbols) are to be considered for the respective UI solution.

UI fragment developers design and implement UI fragments for components at any level of complexity. The scope of these components can range from individual field devices to more complex modules or entire plants. The design of the components includes the definition of the necessary work tasks related to each component in its lifecycle and the information required by the subsequent users to accomplish the tasks. Therefore, for the AAS of the individual components, the UI fragment developers create the UI submodels and the UI fragments to represent the components in a UI solution. Depending on the domain, on typical and special work tasks with regard to the component, and on the planned context of use, different design variants are provided. These variants are realized as individual UI fragments and are described in the AAS using the UI submodel. The description is carried out using suitable properties. For that, developers require appropriate editors that support modeling those properties, which in turn requires properties to be formally specified.

UI solution developers as part of the system integrators create functional UI solutions from any number of available UI fragments. For this purpose, suitable fragments must be selected based on the specific UI solution requirements (cf. Fig. 1) and applied to the solution. Today, these tasks are mostly performed manually. In future, however, system integrators will work more abstractly and with the aid of automatic tools[33]. Instead of creating the UI solution manually, UI solution developers should only have to specify requirements for the UI solution. For the selection of the UI fragments the same vocabulary must be used by the UI submodel and by the requirements model of the UI solution – the *UI fragment properties*. The use of automatic tools requires those properties to be formally and semantically unambiguously specified. The assembly of UI fragments to a UI solution, additionally, requires the implementation technologies used for each fragment to be compatible to each other and to the design tool.

From these considerations, the following property categories for the characterization of UI fragments can be derived:
A) *use case* properties for the description of the purpose of the fragment, B) *context of use* properties for the description of the assumed design constraints, C) *design* properties for the description of design decisions ensuring the UI solution's consistency, and D) *technical* properties that ensure the compatibility of UI fragments and design tools.

2.2 UI Fragment Properties

In this section, we elaborate on the identified property categories by deriving some relevant properties from common requirements for industrial UI solutions, except for the context of use properties that we will focus on in subsequent publications. For better readability, properties are labeled with `monospaced` fonts. Property categories are additionally suffixed with the term `properties`. Due to the high number of properties that can be found for each of these categories, we only want to focus on a few examples.

`Use Case Properties` For the selection of UI fragments for a UI solution, the specific work task particularily must be used as one of the most important criteria for the content and design of a UI fragment[14]. For the purpose of a fine-granular differentiation of UI fragments in terms of content, as demanded in the literature[21; 22], we understand the term *specific work task* as a) the *task type* (also referred to as *task class* or *abstract task*[5; 12; 30]) in combination with b) the *objects* of interest of the UI fragment (also the object of the work task). The use case of the UI fragment results again from the specific work task (a+b) and c) additional parameters for a more detailed determination. An exemplary use case could thus be constructed, as shown in Fig. 2. The task type "Maintenance" and the object "Component X" are indicated as well as the domain "Mechanics" that represents an additional parameter, which, in this case, may correspond to a subsystem of the component (the UI fragment shall support these aspects specifically).

This results in the following properties for the description of the UI use case:
`Supported User Task Type`: The UI fragment denotes one or more types of user tasks that it supports. In the life cycle of a technical plant different task types are relevant with regard to individual field devices, but also with

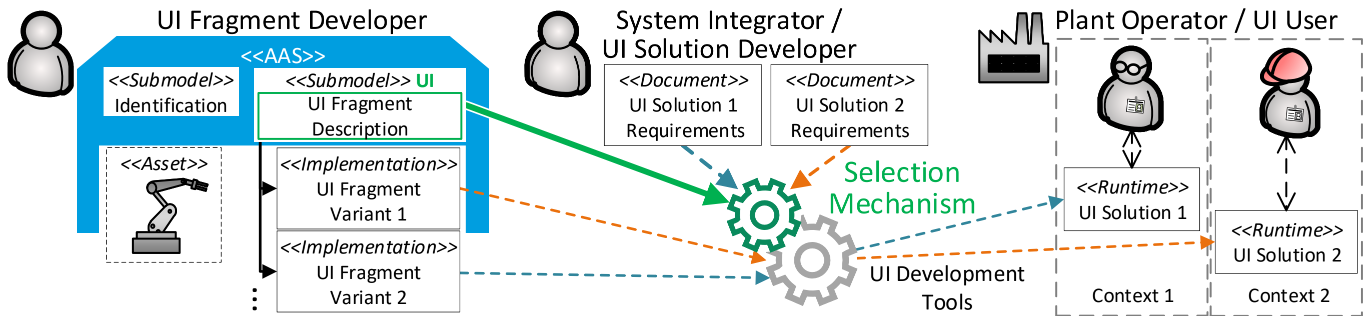

UI Fragment Developer
<<AAS>>
<<Submodel>> Identification
<<Submodel>> UI
UI Fragment Description
<<Asset>>
<<Implementation>> UI Fragment Variant 1
<<Implementation>> UI Fragment Variant 2
System Integrator / UI Solution Developer
<<Document>> UI Solution 1 Requirements
<<Document>> UI Solution 2 Requirements
Selection Mechanism
UI Development Tools
Plant Operator / UI User
<<Runtime>> UI Solution 1
<<Runtime>> UI Solution 2
Context 1
Context 2

Fig. 1. UI stakeholders and their relations to a fragment-based UI design process

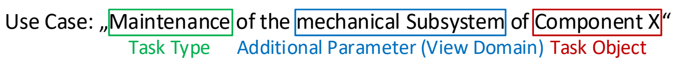

Use Case: „Maintenance of the mechanical Subsystem of Component X"
Task Type
Additional Parameter (View Domain)
Task Object

Fig. 2. Example for the decomposition of a use case.

regard to more complex subsystems and the entire plant as well[29]: Different subsystems must be configured and parameterized before, during, or after integration. They must further be tested during commissioning, monitored, managed, operated, charged, discharged, or evaluated and maintained during operation, taken out of operation, inspected, and repaired in the event of a fault, put back into operation, etc. Regarding the corresponding plant component, each of these task types may require different UI designs, due to different sub-tasks and specific UI contents.

`Represented Entities`: The UI fragment identifies the objects of interest of the user tasks, and thus, the represented objects. In general, the asset (e. g. a process component) of the I4.0 component to which the UI fragment is assigned via the UI submodel in the AAS (see Fig. 1) would be considered as the represented entity, i. e. this property would point to the AAS that contains the UI fragment. In such case, this property can be omitted because the object of the UI fragment is already implicitly known. However, especially within a complex I4.0 component, e. g. a process module, a multitude of different subcomponents in different combinations can be represented in a UI fragment. At the same time, these different fragments possibly are assigned to the same complex I4.0 component and also to the same task type. By means of the `Represented Entities` property, the individual fragments can be distinguished from each other explicitly. This enables the assignment of the fragments to a superordinate AAS of a complex component and the selection of a fragment spcifically according to the user's tasks.

`View Domains`: As one of many other task parameters[30], the view domains property denotes the domain-specific view in which the entities and the structures in between them are displayed. This is decisive for the selection of the UI fragment (see Sect. 5). In addition to the mechanics domain already mentioned, electrical, energy, automation, process or information technology, as well as product- or process-specific disciplines are to be considered.

`Design Properties` In many cases, the organization of the plant operator determines *design guidelines* either locally at the plant level or globally for the entire company. Typically, such guidelines, e.g. for graphical UIs, contain instructions for coloring, for the use of certain symbols, for the arrangement of elements within complex views (order criterion), for structuring the navigation, or for the use of animations for displaying certain process data. The VDI/VDE Guideline 3699-2[36] contains, for example, a color scheme for operator screens or DIN EN ISO 10628[1] specifies symbols for process engineering components. Such guidelines have the purpose to support the creation of consistent UI solutions[7; 8; 10] with regard to the design of the representation and the interaction, taking into account the context of use. However, the characterization of the context of use is not sufficient to ensure consistent UI solutions, since the design decisions regarding the UI appearance and behavior are made within a degree of freedom given by the context of use[39]. Thus, design guidelines are to be taken into account when selecting suitable UI fragments for a UI solution. The following property describes the design guidelines applied to the UI fragment:

`Applied Design Guideline`: If available, applied design guidelines shall be referenced.

If no guidelines are applicable, or if the applied guidelines are not sufficiently specific for the concrete UI fragment, or if certain design decisions were intentionally deviated from the guidelines, the UI fragment must enlist further detailed design properties. E. g. the `Representing Symbol` property which assigns a specific encoding metaphor (e. g. a color, a symbol) to something that is represented within the UI (e. g. an entity, medium, state).

`Technical Properties` Many different process visualization technologies are relevant in industry. These include manufacturer-specific technologies which are usually not standardized. Standardized UI technologies are also common, such as the MTP-HMI or the FDI UI (cf. Sect. 3.1). Furthermore, UIs based on generally available UI frameworks are also common, e. g. based on WPF[38], HTML5[20], etc. Furthermore, formal domain-specific languages such as MARIA[31] or UsiXML[27] are popular in the research community. To describe the technology used for the UI fragment implementation, we have identified the following properties:

`UI Technology Identifier`: A UI fragment must identify the technology in which it was implemented.
`UI Technology Version Supported`: The version of the UI technology used must be identified.

2.3 Remarks on the UI Fragment Properties

An initial identification of properties as we described in the previous section is sufficient to ensure a common vocabulary between UI fragment developers and UI solution developers (see Fig. 1). For the use of the properties in the modeling of the UI submodel, however, according to Sect. 2.1 a computational realization of their specification is necessary. This will be discussed in Sect. 4.

As already suggested, not all properties are always required to describe a UI fragment. For example, the property of the `Represented Entity` can be omitted if the UI fragment is an elementary representation of only the one component that contains the UI-Fragment. Other properties can be considered redundant, e. g. the `Applied Design Guidelines Property` is redundant to all remaining design properties, if the design guidelines already contains these rules. Since this assumption cannot be ensured a priori and especially not, if the design guideline is not known, the redundant properties, however, cannot be ignored. Hence, it seems reasonable to specify and use such properties even if they may be redundant to other properties in order to always support a UI fragment selection tool as best as possible.

Within a UI submodel, but also in the requirements for the UI solution, a statement about the applicable or required value of a UI fragment property must be made (cf. Sect. 3.2). To do so, it is necessary to assign at least one data type to each property. For some properties, primitive data types, such as a string, can be used. This is sufficient, for example, for the UI Technology Identifier being represented by a string. For other properties, enumerated data types can also be specified to simplify the creation of property statements. However, not all property values can be described using primitive data types, complex data types such as references, lists, value ranges, and structured types are also required. The Represented Entities property, as the name implies, should be described as a list of references to the represented entities (e. g. referencing AAS), since a fragment can represent multiple entities and references can be resolved unambiguously. Structured data types can be used whenever multpiple statements need to be made in combination, for example, to describe a representing symbol property which requires to reference the symbol together with the encoded concept.

3. STATE OF THE ART

3.1 UI Fragment Distribution using Components

Regarding existing industrial solutions on the distribution of UI fragments by use of a process component, two standards are worth mentioning: the human machine interface (HMI) of the Module Type Package (MTP) specified by VDI/VDE/NAMUR 2658-2[37] and the Field Device Integration (FDI) package specified by DIN EN 62769-2[16].

FDI provides a UI based on EDDL and WPF that provides different variants for one process component only. These variants are distinguished using a similar property to the supported user task type, but only specifies three different task types. Implicitly, the process component associated to the package is considered as represented entity. Two other properties regarding the context of use are provided: the platform (desktop or mobile) and the user's role (administrator or operator). Considering the technology neutral design of the AAS, it seems not to be useful to specify the UI submodel based on FDI since it depends on a specific platform setup (interpreter and WPF required).

In contrast, the MTP-HMI specification is not limited to any UI technology and provides an extensible UI description that constitutes process mimics for process modules. The technology of which (AutomationML) is very similar to the data modeling approach of the AAS (cf. Sect. 3.2 below). However, this package standard does not support the provision of multiple UI fragments yet. Different variants can only be created during the required transformation step that makes use of properties similar to the represented entity property in order to insert UI Fragments into the mimics. However, the UI submodel specification as presented in Sect. 4 can be used to extend the MTP HMI description.

3.2 AAS Submodel Specification

The available literature on the AAS[3; 4; 32] provides hints regarding the specification of submodels. A representative excerpt of the proposed AAS metamodel concerning the submodel specification is shown in Fig. 3. In the submodel, an arbitrarily nested data structure can be created (by means of the class `StructuralElement`). Hereby, the metamodel is completely generic, and hence, independent from the concrete submodel. For a domain-specific (i. e. submodel-specific) use of these data structures these structural elements must be semantically tagged. For this purpose semantic references[25; 32] (attribute `semantics`) are used that assign a meaning to each structural element and thus ensure an unambiguous interpretation of the data structure. The semantic reference points to an external description of a knowledge unit (also referred to as *concept* – a term agreed upon between parties[23]), e. g. a property specification. The properties of the entities are described by using property statements (class `PropertyStatement`). The property statement assigns a concrete value (attribute `value`) to a specific property which, however, can only have primitive data types up to now. Again, a semantic reference is used to indicate which property is meant.

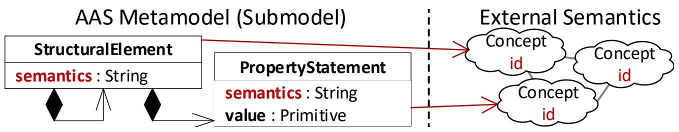


Fig. 3. AAS Metamodel (representative excerpt) regarding the submodel specification and the concept of semantic references (left) using an external semantics specification (right)

Various tools can be used for the external semantics definition: In the industrial context, property catalogs (also known as semantic dictionaries) have been established, such as eCl@ss[17], CDD[26], ETIM[18], and others. Within those catalogs, a series of properties is assigned to classes of entities (e.g. the device class 'pump' comprises the property 'nominal power' of the data type float). For the description of an entity in the AAS, the semantic reference of a structural element would point to the entity class in the catalog. Furthermore, the structural element would contain a series of property statements that in turn point to the property specification in the catalog through the use of their respective semantic references. Hence, an entry in a property catalog can be sufficient to specify the structure and content of the entire submodel, if the submodel characterizes a single entity only by providing a simple list of property statements as specified by a catalog for the entity class.

For more complex submodels, in addition to the individual properties, the structure of the submodel must also be specified in order to establish the meaning of composed structural elements. It has already been suggested to use UML class diagrams[32] for the submodel specification as their elements can be mapped to the AAS metamodel (UML class to `StructuralElement`, UML attribute to `PropertyStatement`, etc.). Other specification tools such as EMF (Eclipse Modeling Framework[34]) software models, XML schemata[19], or ontologies[24] can also be used. Using such tool, the entire semantics of the submodel and its elements can be specified.

4. UI SUBMODEL DESIGN

In this section, we present our proposal for a submodel for the component-based provision of UI fragments. Since several UI fragments are to be described in the UI submodel (see Fig. 1), a complex structure is required. Therefore, the submodel specifications will be represented by using a UML class diagram.

Due to the already suggested mapping of UML to AAS elements (cf. Sect. 3.2), all properties of a UI fragment would need to be specified as attributes of a class that describes the fragment. The identified properties however were also found to be incomplete. Each time when further properties are identified in the future, the UI submodel specification would have to be modified. Therefore, it seems reasonable to use UML for the specification of the submodel structure only and to additionally develop a property catalog as a separate tool that can be extended at any time without changing the specification of the UI submodel.

Therefore, the following design is divided into two parts: First, a suitable submodel structure must be defined (corresponds to left side of Fig. 3) and second, a property catalog (corresponds to right side of Fig. 3) must be developed based on a certain catalog tool.

4.1 Structure of the UI Submodel

Figure 4 shows our proposal for the structure of the UI submodel. It is either realized by using the class `UIFragmentRepository` with several UI fragments or the class `UIFragment` for one fragment only. At first, each fragment consists of at least one `UIFragmentImplementation` object and a `UIFragmentDescription` object that carries applied property statements. However, there are UI implementation technologies available that support adaptation mechanisms. One implementation may thus represent multiple design variants, i. e. multiple UI fragments, at once. Moreover, it is imaginable, not to understand the mere implementation under the term UI fragment, but the representation of a component in a UI. A UI fragment may, thus, be realized by multiple implementations, each relizing a UI fragment variant. Hence, a UI fragment may consist of multiple implementation objects each of which is described by at least one description object. The final decision on how to organize the UI submodel can be made by fragment developers. The only requirement is, that the selection tool (cf. Fig. 1) determines a suitable implementation of a UI fragment for subsequent tools.

The implementation is either provided by a download link in the form of a `URI` (the implementation may be provided from extern) or by direct access to a `File` by means of the AAS (the implementation is shipped with the I4.0 component). Due to the AAS metamodel (cf. Fig. 3), a limitation has to be taken into account regarding the UI fragment properties: Value assignments in form of property statements are only expressed using primitive data types. Sect. 2.3, however, states that complex data types such as lists, structures, value ranges, or references are also required for UI fragment properties. Therefore, the AAS metamodel class `PropertyStatement` cannot be used if a concrete value assignment is to be made for such a property. Instead, a `StructuralElement` has to be used for a property statement that, for example, in case of a list, contains a `PropertyStatement` for each list entry. In the sense of consistent modeling, a `GeneralizedPropertyStatement` that contains `PropertyValue` objects is to be used to describe the properties of a UI fragment until any extension of the AAS metamodel specification is made in future developments. This generalized property statement is able to describe primitive value assignments (classes `NumericValue`, `StringValue`, etc.), but also complex ones (class `StructuredValue`). Compared to the original property statement as shown in Fig. 3, the generalized property statement is also extended in accordance with the requirements in Sect. 2.3 to the effect that value ranges using the `minValue` and `maxValue` references and lists of values using the `value` reference can also be created. As with the original property statement, the generalized property statement is a generic data element without any meaning. Therefore, it also refers to an external property definition which is presented in Sect. 4.2 below by using a semantic reference (attribute `semantics` – cf. Sect. 3.2).

In cases where a component is to be provided with a larger number of UI fragment variants, we expect, also considering the large number of identified properties (see Sect. 2.2), that some fragment descriptions will contain multiple identical property statements and only a few differing statements. Hence, we have introduced the `specialization` reference, allowing `UIFragmentDescription` objects to be nested into each other. Thus, common properties can be extracted into a superordinate description that is not necessarily assigned to an implementation.

4.2 Property Catalog Specification

The most important requirements for property catalogs are the ability to assign unique identifiers (referenceable IDs, see Fig. 3) for the properties specified therein and to ensure the availability of the specification to the modelers and their modeling tools. This includes the modelers of the submodels or the requirements for the UI solution (see Fig. 1) and also the selection tool being part of the UI development tools. The modelers must access the textual representation and explanations of the catalog entries. The

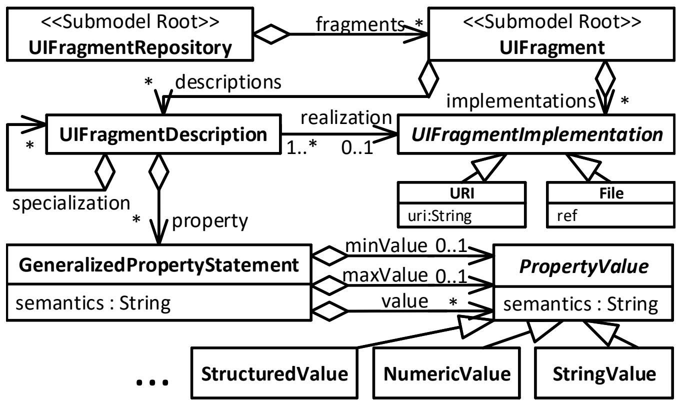


Fig. 4. The proposed AAS UI submodel structure

design tools (e. g. the submodel editor) require access to the formal specification of the data types of each property in order to support the modelers. In the case of UI fragments, these data types must allow the modeling of primitive data types, but also more complex ones, such as lists and structures (see Sections 2.3 and 4.1).

For the realization of a suitable property catalog, several options are available according to Sect. 3.2. Existing catalogs such as the eCl@ss catalog could be extended with the required properties. This however is typically not possible, since the underlying tools (i. e. the catalog metamodels) only allow to specify properties with primitive datatypes (in the case of eCl@ss, see [2]). Some catalog tools are based on the metamodel that is specified by the IEC standard 61360[26], which also does not support complex datatypes in general despite supporting ranges, lists, etc.

Therefore, in the context of the development of the UI submodel, a separate catalog tool must be developed and, based on this, the necessary catalog for UI fragment properties. For the catalog tool, we decided to specify a metamodel based on EMF[34], since it allows us to quickly generate an appropriate catalog editor and other tools like a generator for an online documentation for publishing our catalog instances. The concrete specification of this catalog tool, however, is not the focus of this contribution and will hence not be discussed in detail.

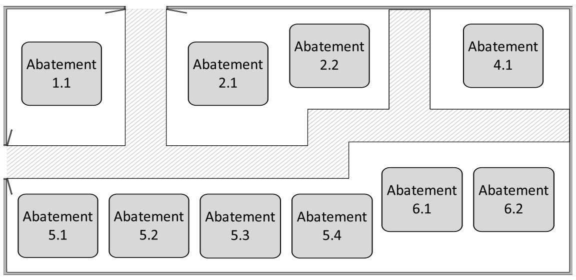
In order to enable the use of the identified properties (see Sect. 2.2) in a UI submodel, we created a *UI-Frament-Properties* catalog. Since we expect certain aspects of this catalog also to be useful in other contexts, we extracted those aspects to separate catalogs. Thus, we created a *Tasks* catalog specifying some task types that are used in the `Supported User Task Type` property (see Sect. 2.2.1) and a *Domains* catalog specifying some industrial domains that are used in the `View Domain` property. Both additional catalogs may, e. g., also be useful in the domain of task planning and modeling. All catalogs can be reviewed online: `https://agtele.eats.et.tu-dresden.de/uimdf`.

5. CASE STUDY

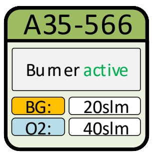
In a first case study we want to demonstrate that UI fragments of process components can be used to create complex representations according to the plant structure and varying use cases that cause different fragments to be selected. In cooperation with the company *DAS Environmental Experts GmbH* (`https://www.das-ee.com`), we have carried out a component-based UI design for an off-gas abatement system of the company. Hereby, the task of an abatement unit is to clean a process gas so that it can be released into the atmosphere. The aim of of the case study was to design screens for a monitoring system, which would be used mainly at locations where multiple abatement units are used. The screens are based on a site plan (see Fig. 5a), in which various block representations of the units are to be inserted. Among others, the units provide UI fragments for nominal operations (see Fig. 5b), for maintenance planning (see Fig. 5c), and for assessing the consumption of electrical energy (see Fig. 5d). For the commissioning and decommissioning of a plant, the site plan should also show the connection of the units to their respective peripherals. Taking into account that there are not always 1:1 relationships between peripheral components and an abatement unit, rather complex mimics have to be created. Fig. 5e shows a case in which several units are connected to one valve of the gas supply train. If the gas supply is to be interrupted, for example to replace a unit, this may not be possible until all units on this segment have been switched off. By use of the view domain property, the same task can be carried out for different subsystems. For the fault diagnosis of an individual unit, however, different supply systems (water, nat. gas, el. power) must be taken into account at the same time (see Fig. 5f). First, the three UI fragment versions b, c and d can be assigned to the AAS of the abatement unit and described by means of some properties from Sect. 2.2 (see Tb. 1). The overview pictures 5e and 5f created based on the generic site plan 5a as well as the empty site plan itself can also be understood as UI fragments, too, and thus, described in a superordinate AAS (see Tb. 2). The selected UI fragment properties use references (symbol →) to the aforementioned separated catalogs extensively. Those references are realized using strings with resolvable identifiers as values in the submodel. Due to the definition of the property in the catalog, the identifiers can be validated so that the intended reference target is checked. The submodel editor provides support to the modeler by proposing possible values. Hence, the mentioned requirements (cf. Sect. 2) for the modeling tools and the property specification are met. In this initial step of a broader case study, the applicability of the identified properties was demonstrated. However, the demonstration of an automatic selection of the UI fragments, the detailed demonstration of the UI submodel modeling, and the UI solution requirements modeling remain for future works.

Table 1. Prop. Statements Fragments 5b, 5c, 5d. Symbols definition: ① Supported User Task Type; ② Represented Entities; ③ View Domain; ④ Applied Design Guideline; → reference to external object/definition

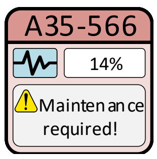
Prop.	Statements 5b	Statements 5c	Statements 5d
①	→ supervision	→ monitoring	→ supervision
③	∅	∅	→ electrical energy
④	[DAS UI Design Guideline Identifier]		



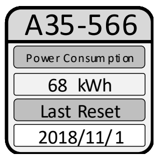
(a) Site plan



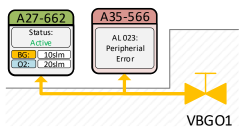
(b) Block State



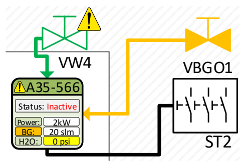
(c) Block Health



(d) Block Power



(e) Gas Supply Train



(f) Unit Periphery

Fig. 5. Site plan and different UI fragments

Table 2. Prop. Statements Fragments 5a, 5e, 5f. Symbol definition, see Tb. 1.

Prop.	Statements 5a	Statements 5e	Statements 5f
①	→ navigation (in the plant)	→ commissioning	→ fault analysis
②	→ floor → DAS Abatement Type	→ A27-662 → A35-566 →VBG01	→ A35-566 → VW4 →VBG01 →ST2
③	∅	→ gas supply	→gas supply →water supply →el. power
④	∅	[DAS UI Design Guideline Identifier] [DIN EN ISO 10628 Identifier] . . .	

6. CONCLUSIONS

In this contribution, we have presented a first version of a UI submodel specification for the I4.0 Component. Using this model, several UI fragments can be provided for each component and described by a list of formally specified properties. In the future, the properties introduced in this article can be used to identify and select an appropriate variant using an automatic tool for inserting the fragment into a UI solution. In order to be able to use the properties within the generic UI submodel in an unambiguous way, we have created several property catalogs. The definition and description of UI fragments has already been tested in a first case study for an industrial application. Thus, the applicability of our approach has been demonstrated. The metamodel of the UI submodel, the catalogs and related tools have been combined to the *UI Meta Description Framework* (UIMDF) which we intend to publish in the repositories of our institute (`https://gitlab.com/tud-ifa`) in the future.

Yet more research and development remains to be done: We are not expecting additional effort for UI fragment developers when integrating the use of fragment properties into existing UI design workflows since the UI properties can simply be seen as a formalization of the UI design constraints which, in a proper UI design process, have to be defined anyway. However, creating and maintaining multiple UI fragments that may be needed in PnP scenarios for a single component can be a resource-intensive process. Due to the large number of available properties, the problem of variant explosions quickly arises. Here, in addition to the introduced feature of abstracting UI fragment descriptions (cf. Sect. 4.1), further approaches have to be investigated, for example in the field of model-based user interface development[9] (MBUID). In this area, approaches already exist that allow the adaptation of a UI fragment implementation to the current context of use through parameterization[21]. For this purpose, properties must also be specified describing the intended context of use of a UI fragment. Unfortunately, there are very few publications that present formally specified context properties[28]. Therefore, in future works, appropriate properties have to be specified and additional catalogs have to be created that comprise properties for the description of the context of use. In addition, the description of the requirements of the UI solution and the realization of an automatic selection algorithm are to be discussed. Because of the complex data types that can be used in property statements, the comparison of property statement and requirement becomes a non-trivial problem. Furthermore, the generation process of the UI solution has to be considered, which may lead to the specification of further UI fragment properties (e. g. for parameterizing the UI fragment). Finally, we want to investigate and refine the already identified properties in further case studies.

ACKNOWLEDGEMENTS

We would like to thank the company *DAS Environmental Experts GmbH* for the courtesy of sharing their insights and *Maksym Lebedyk* for his work on the case study.

REFERENCES

[1] (2001). *Fließschemata Für Verfahrenstechnische Anlagen Allgemeine Regeln (ISO 10628:1997) Deutsche Fassung EN ISO 10628:2000*. DIN EN ISO 10628. Beuth Verlag GmbH, Berlin.

[2] (2019). Property - wiki.eclass.eu. URL `http://wiki.eclass.eu/wiki/Property`.

[3] 4.0, P.I. (2019). Verwaltungsschale in der Praxis. Technical report, Plattform Industrie 4.0, VDI/VDE-GMA.

[4] AG Referenzarchitekturen, Standards und Normung (2016). Struktur der Verwaltungsschale.

[5] Akiki, P.A., Bandara, A.K., and Yu, Y. (2016). Engineering Adaptive Model-Driven User Interfaces. *IEEE Transactions on Software Engineering*, 42(12), 1118–1147.

[6] Bedenbender, H., Bentkus, A., Epple, U., et al. (2017). Industrie 4.0 Plug-and-Produce for Adaptable Factories: Example Use Case Definition, Models, and Implementation. Technical report.

[7] Bevan, N. (2001). International standards for HCI and usability. *International Journal of Human-Computer Studies*, 55(4), 533–552.

[8] Butz, A. and Krüger, A. (2017). *Mensch-Maschine-Interaktion.* Walter de Gruyter GmbH, Berlin/Boston, 2 edition.

[9] Calvary, G., Coutaz, J., Thevenin, D., et al. (2003). A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers*, 15(3), 289–308.

[10] Denis, C. and Karsenty, L. (2004). Inter-Usability of Multi-Device System - A Conceptual Framework. In *Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces.* J. Wiley, Hoboken, NJ.

[11] Deutsches Institut für Normen (2016). *Referenzarchitekturmodell Industrie 4.0 (RAMI4.0).* DIN SPEC 91345. Beuth, Berlin.

[12] Diaper, D. and Stanton, N.A. (2004). Wishing on a sTAr: The Future of Task Analsysis. In *The Handbook of Task Analysis for Human-Computer Interaction*, 603–619. Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey.

[13] DIN-NAErg (2011). *Ergonomie - Genereller Ansatz, Prinzipien und Konzepte (ISO 26800:2011); Deutsche Fassung EN ISO 26800:2011.* DIN EN ISO 26800. Beuth Verlag GmbH, Berlin.

[14] DIN-NAErg and DIN-NIA (2011). *Ergonomie Der Mensch-System-Interaktion - Teil 210: Prozess Zur Gestaltung Gebrauchstauglicher Interaktiver Systeme.* DIN EN ISO 9241-210. Beuth, Berlin.

[15] DIN-NAErg and DIN-NIA (2017). *Ergonomie der Mensch-System-Interaktion – Teil 11: Gebrauchstauglichkeit: Begriffe und Konzepte (ISO 9241-11.2:2016); Deutsche und Englische Fassung prEN ISO 9241-11:2016.* 9241-11. Beuth Verlag GmbH, Berlin.

[16] DIN/VDE-DKE (2016). *Feldgeräteintegration (FDI) – Teil 2: FDIClient (IEC 62769-2:2015); Englische Fassung EN 62769-2:2015.* DIN EN 62769-2. Beuth Verlag GmbH, Berlin.

[17] ecl@ss e.V. (2019). The eCl@ss-Standard. URL `https://www.eclass.eu/standard.html`.

[18] ETIM International (2019). ETMI International. URL `http://community.etim-international.com`.

[19] Fallside, D.C. and Walmsley, P. (eds.) (2004). *XML Schema Part 0: Primer Second Edition.* W3C Recommendation. W3C.

[20] Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and Sangwhan, M. (eds.) (2017). *HTML 5.2.* W3C Recommendation. W3C.

[21] Freund, M., Martin, C., and Braune, A. (2016). A Library System to Support Model-Based User Interface Development in Industrial Automation. In M. Kurosu (ed.), *Human-Computer Interaction. Theory, Design, Development and Practice. HCII 2016.*, volume 9731 of *Lecture Notes in Computer Science*, 476–487. Springer, Cham.

[22] Gorecky, D., Schmitt, M., Loskyll, M., and Zuhlke, D. (2014). Human-machine-interaction in the industry 4.0 era. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, 289–294. IEEE, Porto Alegre RS, Brazil.

[23] Gruber, T.R. (1995). Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 43(5), 907–928.

[24] Guarino, N., Oberle, D., and Staab, S. (2009). What Is an Ontology? In *Handbook on Ontologies*, 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg.

[25] Hildebrandt, C., Scholz, A., Fay, A., et al. (2017). Semantische Allianz 4.0: Semantische Inhalte für Industrie 4.0. In *Automation 2017*, volume VDI Berichte 2293, 262. VDI Verlag GmbH, Düsseldorf.

[26] International Electrotechnical Commission (2019). IEC 61360 - Common Data Dictionary (CDD - V2.0014.0016). URL `https://cdd.iec.ch/cdd/iec61360/iec61360.nsf`.

[27] Limbourg, Q., Vanderdonckt, J., Michotte, B., et al. (2004). Usixml: A user interface description language supporting multiple levels of independence. *Engineering Advanced Web Applications*, 325–338.

[28] Loitsch, C., Weber, G., Kaklanis, N., et al. (2017). A knowledge-based approach to user interface adaptation from preferences and for special needs. *User Modeling and User-Adapted Interaction*, 27(3-5), 445–491.

[29] NAMUR-AF1-AK NA 35 (2019). *PLT-Planung und -Abwicklung in der Prozessindustrie.* Number NA 35 in NAMUR Arbeitsblatt. NAMUR, Leverkusen.

[30] Paternò, F. (2004). ConcurTaskTrees: An Engineered Notation for Task Models. In D. Diaper and N.A. Stanton (eds.), *The Handbook of Task Analysis for Human-Computer Interaction*, 483–501. Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey.

[31] Paternò, F., Santoro, C., and Spano, L.D. (2009). MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. *ACM Transactions on Computer-Human Interaction*, 16(4), 1–30.

[32] Plattform Industrie 4.0 (2018). Verwaltungsschale im Detail - Part 1 - The exchange of information between partners in the value chain.

[33] Schmidt, M., Löwen, U., Kalhoff, J., et al. (2016). WFF - Wandlungsfähige Fabrik - Langfassung.

[34] Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0.* Addison-Wesley Professional, 2nd edition.

[35] Trenkle, A. and Furmans, K. (2016). Der Mensch als Teil von Industrie 4.0: Interaktionsmechanismen bei autonomen Materialflusssystemen. In B. Vogel-Heuser, T. Bauernhansl, and M. ten Hompel (eds.), *Handbuch Industrie 4.0 Bd. 3: Logistik*, 45–59. Springer, Berlin / Heidelberg, 2 edition.

[36] VDI/VDE-GMA (2014). *Prozessführung mit Bildschirmen: Grundlagen.* VDI/VDE 3699-2. Beuth Verlag GmbH, Berlin.

[37] VDI/VDE-GMA (2018). *Automatisierungstechnisches Engineering modularer Anlagen in der Prozessindustrie - Modellierung von Bedienbildern.* Number 2658-2 in VDI/VDE/NAMUR-Richtlinien. Beuth Verlag GmbH, Berlin.

[38] Wikipedia (2019). Windows Presentation Foundation.

[39] Zühlke, D. (2012). *Nutzergerechte Entwicklung von Mensch-Maschine-Systemen.* Springer Berlin Heidelberg, Berlin, Heidelberg.