IEC PAS 62883

Edition 1.0 2014-03

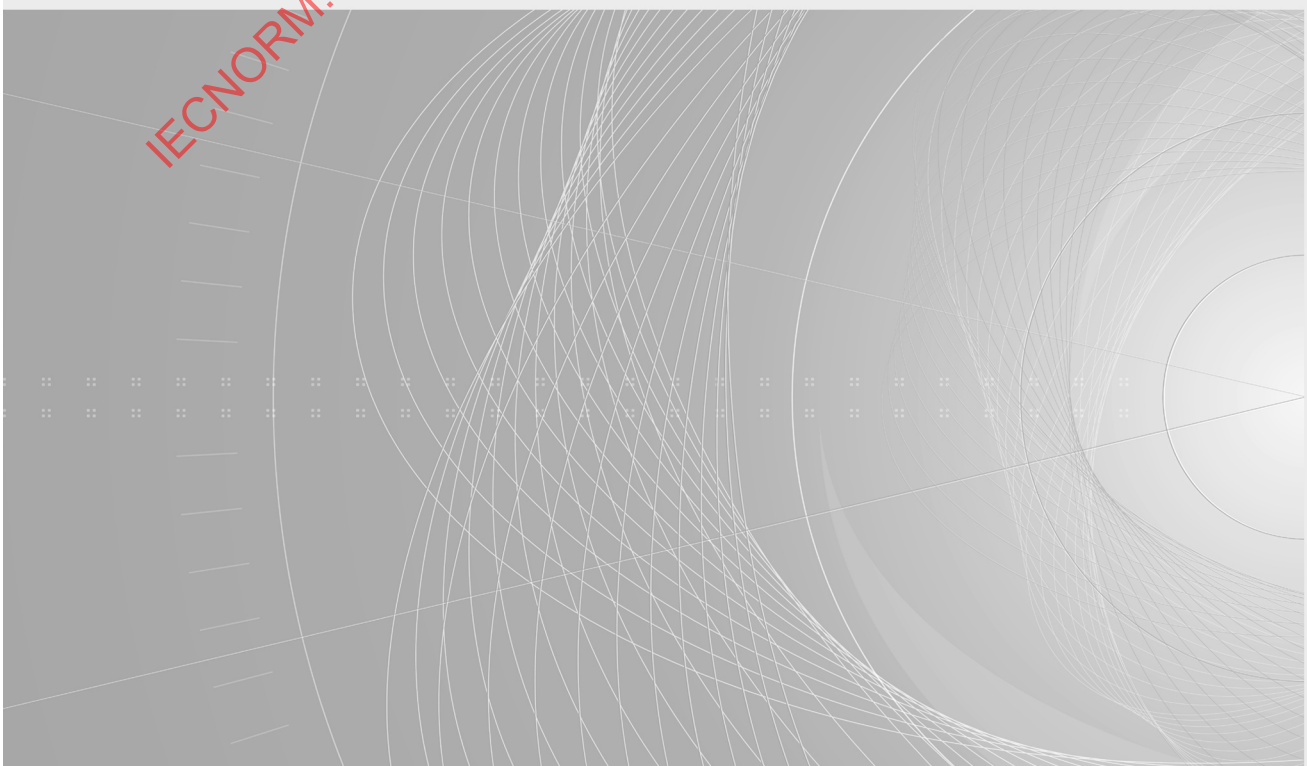
# PUBLICLY AVAILABLE SPECIFICATION

## PRE-STANDARD





**The universAAL framework for user interaction in multimedia AAL spaces**



IEC PAS 62883:2014-03(en)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

**About the IEC**
The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

**About IEC publications**
The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

**IEC Catalogue - webstore.iec.ch/catalogue**
The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

**IEC publications search - www.iec.ch/searchpub**
The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, replaced and withdrawn publications.

**IEC Just Published - webstore.iec.ch/justpublished**
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and also once a month by email.

**Electropedia - www.electropedia.org**
The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in 14 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

**IEC Glossary - std.iec.ch/glossary**
More than 55 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

**IEC Customer Service Centre - webstore.iec.ch/csc**
If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

# IEC PAS 62883

Edition 1.0 2014-03

# PUBLICLY AVAILABLE SPECIFICATION

## PRE-STANDARD



colour inside

**The universAAL framework for user interaction in multimedia AAL spaces**

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE V

ICS 13.180; 33.160

ISBN 978-2-8322-1452-7

**Warning! Make sure that you obtained this publication from an authorized distributor.**

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

# THE UNIVERSAAL FRAMEWORK FOR USER INTERACTION IN MULTIMEDIA AAL SPACES

## FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

A PAS is a technical specification not fulfilling the requirements for a standard, but made available to the public.

IEC-PAS 62883 has been processed by IEC technical committee 100: Audio, video and multimedia systems and equipment.



| The text of this PAS is based on the following document: | This PAS was approved for publication by the P-members of the committee concerned as indicated in the following document |
|---|---|
| **Draft PAS** | **Report on voting** |
| 100/2189/PAS | 100/2228/RVD |

Following publication of this PAS, which is a pre-standard publication, the technical committee or subcommittee concerned may transform it into an International Standard.

This PAS shall remain valid for an initial maximum period of 3 years starting from the publication date. The validity may be extended for a single period up to a maximum of 3 years, at the end of which it shall be published as another type of normative document, or shall be withdrawn.

**IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.**

# INTRODUCTION

Ambient Assisted Living (AAL) strives to ensure the independence, safety, wellbeing and autonomy of users by using ICT, including multimedia systems and equipment and audio / video communication, for creating intelligent living environments that react to the needs of users by providing relevant assistance. Such intelligent environments can be labelled as AAL Spaces, which are characterized by a number of devices that can be stationary, mobile or embedded within other objects. Multiple users can find themselves in an AAL space simultaneously, possibly moving around within the AAL space, and entering and leaving it dynamically. These characteristics introduce new challenges when it comes to handling interaction with users in AAL spaces.

With the assumption that people are surrounded by highly distributed systems of networked interactive devices, AAL intensifies the paradigm shift from Human-Computer Interaction (HCI) to Human-Environment Interaction (HEI). One of the main challenges of HEI is to keep the multiplicity of functional units hidden to humans while making the functionality provided by them easily available based on natural ways of interaction. Instead of controlling each device separately, users should be able to interact with a whole device ensemble as one single unit and articulate goals instead of looking for functionality at the level of each single device separately (Figure 1).

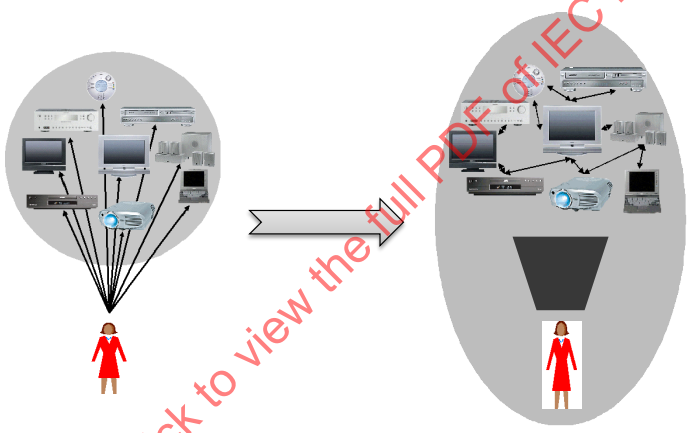


**Figure 1 – Paradigm shift from HCI to HEI**

Another important challenge for designers and developers of systems in AAL spaces is that interaction with applications can take place through a variety of devices at different locations with different capabilities in terms of serving a single user privately or not, supported modalities, modality-specific parameters such as screen size and resolution, power consumption, etc., which implies the need in AAL spaces to logically separate the application layer from the presentation layer (Figure 2).

Consequently, applications have to use abstract user interfaces that are device-, modality-, and layout-neutral and allow to postpone the rendering of the user interface to the execution-time, which makes it possible to interact with users in a personalized and situation-aware way. The separation of concerns also facilitates the creation of clean programming interfaces based on an open and flexible architecture that have to enable the plug-and-play of both applications and user interaction handlers (UI handlers), and allows UI handlers to serve arbitrary applications.

The resulted openness complements the openness supported by IEC 62481-2 that enables the sharing of multimedia content and streams within an ensemble of devices. It adds the

perspective of *sharing the input and output channels provided by those devices*[1] to the DLNA perspective of content sharing.

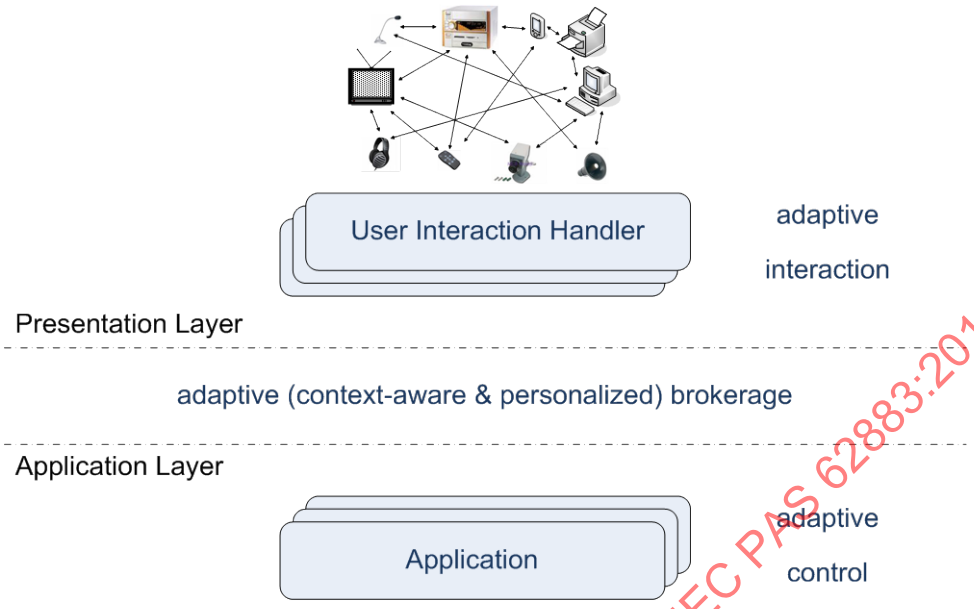



**Figure 2 – logical separation of application and presentation layers**



[1] This understanding of the term I/O channel is based on the actual roles of devices that enable interaction with human users: a display provides a visual output channel, a loudspeaker, an audio output channel, and a microphone, an audio input channel.

# THE UNIVERSAAL FRAMEWORK FOR USER INTERACTION IN MULTIMEDIA AAL SPACES

## 1 Scope

This Publicly Available Specification (PAS) specifies a framework for adaptive handling of explicit interaction among humans and AAL spaces. This is based on a differentiation between explicit and implicit interaction as a consequence of the paradigm shift from Human-Computer Interaction to Human-Environment Interaction, further explained in the definition of the latter term.

As a framework, a main subject matter of the specification is the identification of relevant areas for further standardization, thereby also looking at the interrelationships among the identified areas. The PAS also provides a first extensible specification in some of those areas.

The proposed UI framework has been derived from the logical separation of application and presentation layers as depicted by Figure 2, and encompasses the following elements (Figure 3):

- Analysis of the relationships between UI handlers and I/O devices without specifying possible languages, models, or abstract APIs for interaction with these devices, as there are certain international standardization activities that go in this direction[2];
- the language and model for describing application-specific dialogs / user interfaces as part of UI requests made by applications to the UI framework;
- the adaptation concept and parameters needed to achieve adaptive UI and the way they affect UI requests; and
- Protocols used by the UI framework to broker between UI handlers and applications as pluggable components.



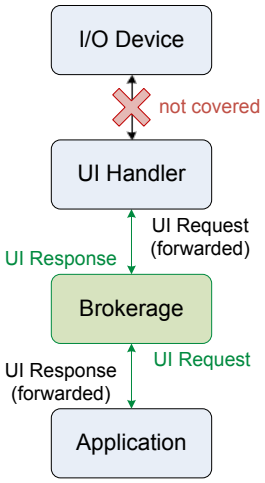



**Figure 3 – The scope of the specified UI framework marked by the green colour**

2 For example [3] on representing user input coming from input devices.

## 2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 62481-2, *Digital living network alliance (DLNA) home networked device interoperability guidelines – Part 2: DLNA media formats*

ISO/IEC Guide 71:2001, *Guidelines for standards developers to address the needs of older persons and persons with disabilities*

ISO 9241-11:1998, *Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability*

ISO 9241-110:2006, *Ergonomics of human-system interaction - Part 110: Dialogue principles*

## 3 Terms, definitions and abbreviations

For the purposes of this document, the following terms and definitions apply.

### 3.1 Terms and definitions

**3.1.1**
**ambient assisted living**
**AAL**
products, services, environments and facilities used to support those whose independence, safety, wellbeing and autonomy are compromised by their physical or mental status

Note 1 to entry: In this PAS, AAL refers to the usage of ICT, including multimedia systems and equipment and audio / video communication, for creating intelligent living environments that react to the needs of users by providing relevant assistance.

[SOURCE: ISO/IEC GUIDE 71:2001]

**3.1.2**
**AAL service user**
person who interacts with an AAL system or is connected with an AAL system

**3.1.3**
**AAL space**
denotes a smart environment that provides AAL services to its users

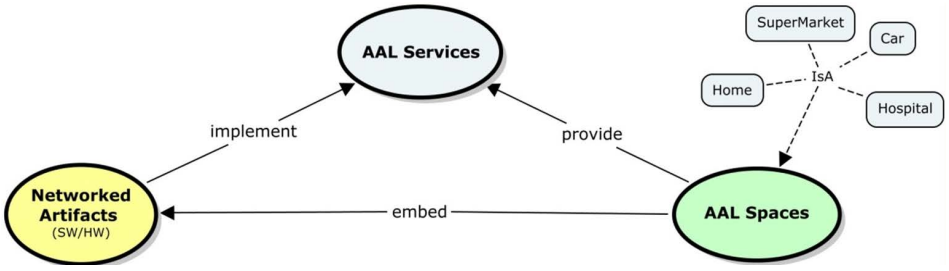



**Figure 4 – The notion of AAL spaces**

**3.1.4**
**channel**
denotes the bridging passage that smart environments need between the physical world and the virtual realm (Figure 5). Channels are realized by devices. Depending on the kind of channel opened, a channel might be called a sensing channel (realized by sensors), an acting channel (realized by actuators), an input channel (realized by microphones, keyboards, etc.), or an output channel (realized by displays, loudspeakers, etc.). The latter two types of channels might be referred to as I/O channels

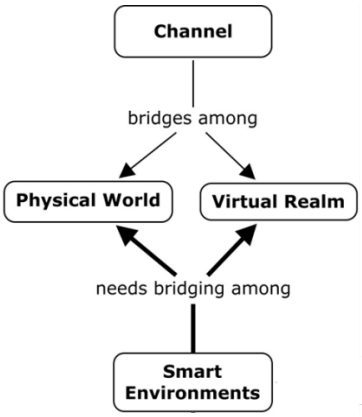



**Figure 5 – The need of smart environments to utilize channels for bridging between the physical world and the virtual realm**

**3.1.5**
**human-environment interaction**
interaction in smart environments is generally divided into two major areas: implicit and explicit interaction

Implicit interaction is mostly about using sensing channels for observation of happenings, with or without involvement of humans, in order to recognize in the background relevant situations to which the environment might be able to react in a desired way.

Explicit interaction, on the contrary, is about situations in which a human user seeks the dialog with the environment or vice versa, for instance when the user instructs that the brightness of the TV should be increased or when the environment notifies the user that it is time to take a certain medicine. Explicit interaction takes place by utilizing input and output channels provided by I/O devices.

**3.1.6**
**I/O device**
an abbreviation for input and / or output device. A device that provides an input and / or output channel for facilitating explicit interaction between a smart environment and its human users. Input devices, such as a microphone, a keyboard, or a mouse, can capture an instruction or response that is provided by a human user and represent it in terms of data in the virtual realm (Figure 6). Upon receive of data within the virtual realm that is intended to be presented to human users, output devices, such as displays and loudspeakers, can make it perceivable to the addressed humans

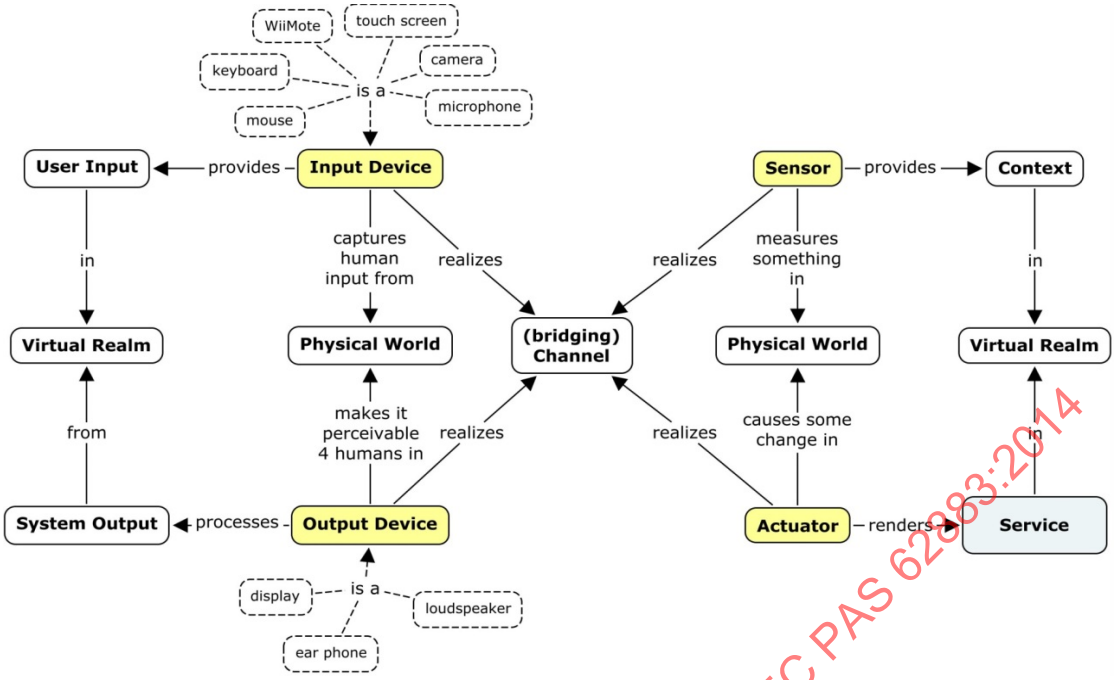



**Figure 6 – The role of devices in realizing bridging channels**

**3.1.7**
**multimodal UI handler**
denotes UI handlers that perform the interaction by using multiple channels in parallel, possibly with a hybrid mix supporting different modalities

**3.1.8**
**smart environment**
denotes an environment centred on its human users in which a set of embedded networked artefacts, both hardware (HW) and software (SW), collectively realize the paradigm of Ambient Intelligence, mainly by providing for context-awareness and personalization, adaptive reactivity, and anticipatory pro-activity

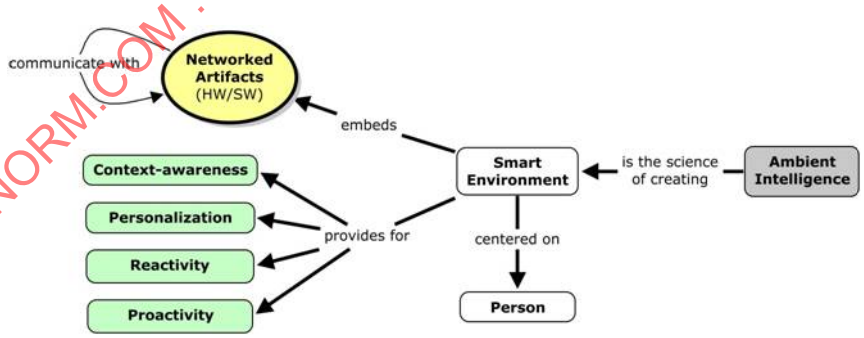



**Figure 7 – The notion of a smart environment**

**3.1.9**
**user**
person who interacts with the product, service or environment

**3.1.10**
**user interface**
all components of an interactive system (software or hardware) that provide information and/or control functions for the user to accomplish specific tasks with the interactive system

### 3.2 Abbreviation

| | |
|---|---|
| AAL | Ambient Assisted Living |
| API | Application programming interface |
| Cf | confer |
| DLNA | Digital Living Network Alliance |
| e.g. | for example |
| etc. | et cetera |
| GUI | Graphical user interface |
| HCI | Human-Computer Interaction |
| HEI | Human-Environment Interaction |
| HTML | Hypertext Markup Language |
| HW | Hardware |
| ICT | Information and communications' technology |
| i.e. | id est, that is to say |
| I/O channels | Input/Output channel |
| OWL | Web Ontology Language |
| RDF | Resource Description Framework |
| RM | Resource Manager |
| SW | Software |
| UI | User Interaction |
| UI model | User Interaction model |
| UIML | User Interface Markup Language |
| uAAL | universAAL |
| UPNP | Universal Plug and Play |
| W3C | World Wide Web Consortium |
| XForms | W3C Standard at http://www.w3.org/TR/xforms/ |
| XML | Extensible Markup Language |
| XPath | XML Path Language; W3C Standard at http://www.w3.org/TR/xpath/ |

## 4 The Specification of the universAAL UI Framework

### 4.1 Overview

As depicted in the Introduction to this PAS, the universAAL UI Framework for Multimedia AAL spaces is based on separating the application layer from the presentation layer by introducing (1) a new type of software components called UI handlers and (2) a brokerage mechanism between the application and presentation layers. The goal is to achieve an open system in which arbitrary applications as well as UI handlers can be plugged in independently from each other while enabling applications to benefit from support for multimodal interaction both by the framework itself and by the UI handlers:

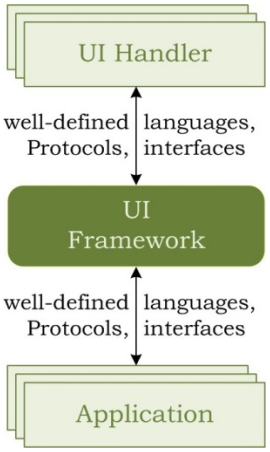



**Figure 8 – An open system for plugging in applications and UI handlers independently from each other**



## 4.2 Analysis of the relationships between UI Handlers and I/O Channels

UI handlers are seen responsible for handling requests for interacting with human users. To do so, they need to utilize the available I/O channels but are not necessarily supposed to take over the binding and management of those channels. On the other side, the "manager" of a single input or output channel is usually not able to handle the whole of a UI request sent by an application because applications often wait for user response in the context of some information to be presented to the user; as a result. At least one output channel and one input channel are supposed to be utilized simultaneously by the same UI handler in order to be able to interpret user input in the context of the output presented to the user.

Therefore, it is important to have a more precise look at the relationship between UI handlers and I/O channels: Section 3.1 states that channels are realized by certain devices but it does not say anything about their binding and availability in the virtual realm. In the concept map in Figure 9, those definitions are extended by introducing the concept of Channel Binding; the same concept can be used for both input and output channels:

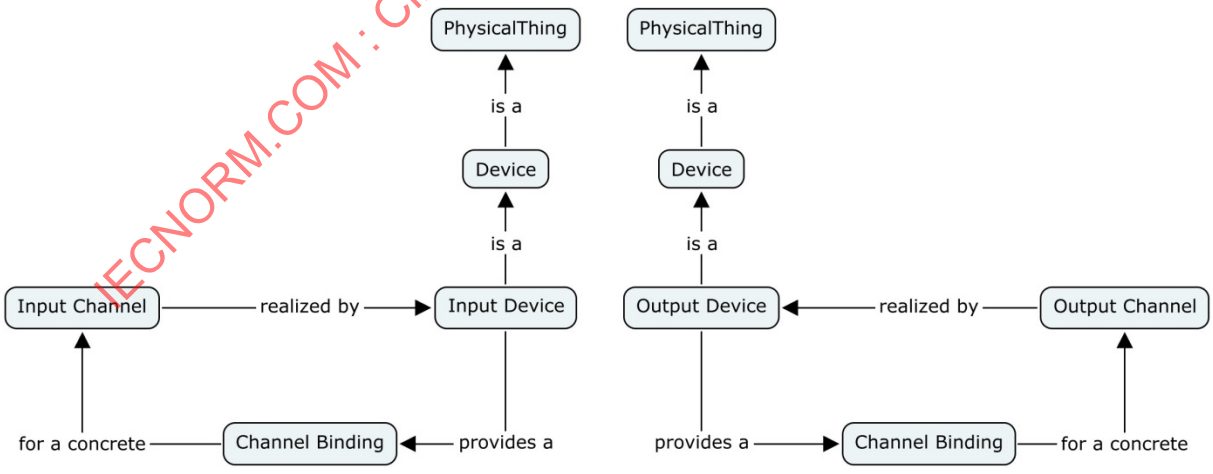



**Figure 9 – Channel binding by I/O devices**

The device that realizes a channel has to have an integrated solution for providing access to the channel. This might be very low-level, at hard- and firmware level to exchange certain bits and bytes through a physical connector interface (called *Embedded Binding)* or already software using higher-level protocols and abstractions (called *Driver)*. Drivers might be provided by third-party and / or run on third-party devices. Like Figure 10 denotes, a driver might wrap another driver in order to comply with higher-level abstractions (e.g. a “UPNP driver”) for binding a special-purpose device that is not readily “UPNP-aware” usually uses

"Embedded Bindings" or "Legacy Drivers" for providing wrappers that interact at the UPNP level of abstraction.

In particular, the higher-level abstractions might only make sense in the context of a framework created for certain purposes (cf. a windowing system that uses a mouse driver and already interprets the mouse events in the context of the bunch of objects in the framework).

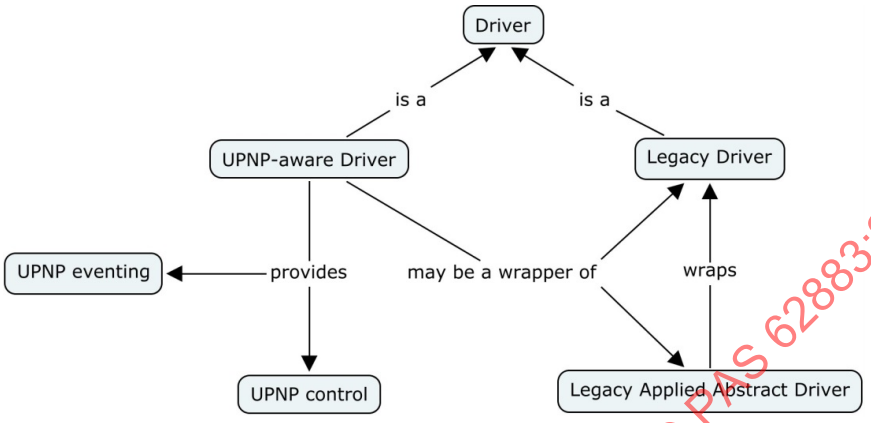



**Figure 10 – The notion of a driver with the case of a UPNP-aware driver[3]**

Similar to the UPNP-aware driver, a universAAL-aware driver should use the API of the universAAL middleware in order to provide related data and functionality in the universAAL realm[4] (Figure 11):

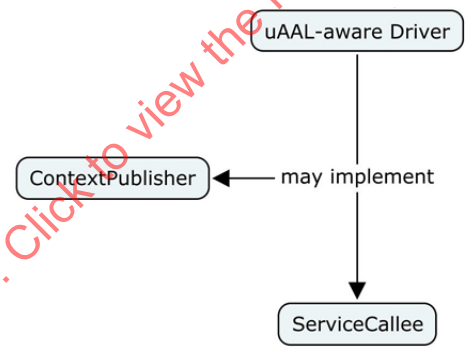



**Figure 11 – The case of a universAAL aware driver**

Like Figure 12 denotes, the development of a UI handler can be based on any of the available alternatives.



---

3 A driver that is UPNP-aware, provides data to the UPNP realm by using the UPNP eventing mechanisms and it renders services in the UPNP realm by using the UPNP control mechanisms.

4 The universAAL middleware provides for sharing data and functionality using the abstraction of virtual communication buses that broker messages between attached components; there are three buses, each responsible for a specific class of brokerage cases: the context bus works on a publish-subscribe base and is responsible for the brokerage of events, the service bus works on a request-response base and brokers services, and finally the UI bus (subject to discussion in Section 4.5.2) that brokers requests for interaction with human users to UI handlers. The three buses of the universAAL middleware are quite in analogy to the building blocks "Eventing", "Control" and "Presentation" in the common architecture of UPNP. To publish events on the context bus, a component has to extend an interface called "Context Publisher", and to provide services on the service bus, it has to extend "Service Callee".

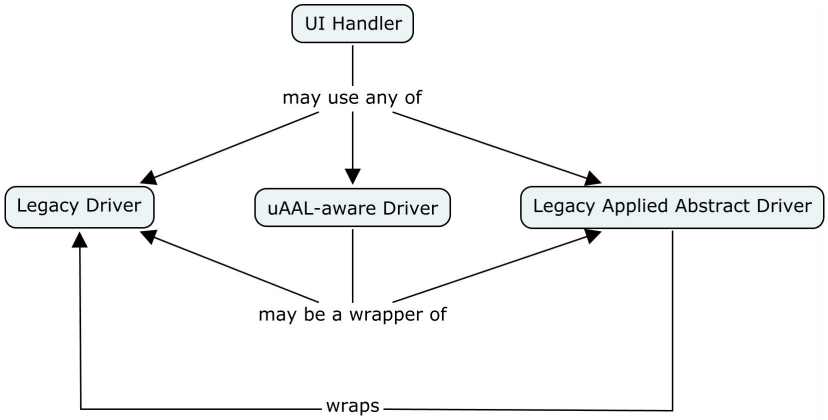



**Figure 12 – Possible relationship between UI handlers and drivers**

An input or output channel can be associated with a certain location (the location of the device that realizes the channel), a modality, and a privacy level[5]. These are important characteristics when it comes to context-aware and personalized selection of an appropriate UI handler, which is supposed to be a crucial task of the UI framework.

There is an analogy between UI handlers and Web browsers ([4]). The specific characteristics of UI handlers in comparison to Web browsers are:

- UI handlers in AAL spaces might use modern middleware to utilize I/O channels distributed in an AAL space instead of being limited to only locally available drivers;
- Compared to the Web, the UI framework for AAL spaces might move easier beyond HTML, which is not really modality- & layout-neutral, and make use of the results of activities, that target the development of applications beyond the browsing of Web pages.

### 4.3 Dialog descriptions

The most important part of UI requests is the description of the dialog that an application is asking to be held with a certain user. Such a description shall be device-, modality- and layout-independent in order to guarantee a clear separation between application logic, on one side, and presentation mechanisms used by the UI handlers available in a given AAL space, on the other side. There are several alternative specifications relevant for dialog descriptions; this document builds on XForms 1.1. XForms scripts are XML documents basically formed from two parts:

- a set of form Controls that define the structure of the form as to be presented to the addressed human user; and
- a set of Model elements that mainly deal with the data involved in the intended interaction from the viewpoint of the application.

The latter includes the data structures and types relevant for the form data as well as instance data, such as hidden data or any initial values to be used when rendering the form or hidden data. Form controls are linked with the XML elements in the model part via XPATH expressions[6]. In this way, the interpreter will know if any initial value is associated with a form control at hand and which restrictions have to be applied to possible related user input.

The application of XForms in the universAAL UI Framework is not one-to-one due to the fact that the UI framework specified in this document is part of a more complete specification on

---

5 Privacy-level of a channel denotes its appropriateness for private communication when several people are in the location where interaction takes place.

6 See http://www.w3.org/TR/xpath/. XPATH expressions can be used to refer to the XML elements and attributes in an XML document.

application-to-application interoperability within AAL spaces, in which data sharing plays a substantial role. For this reason, universAAL had to rely on the Semantic Web[7] specifications and used the Resource Description Framework[8] (RDF) as the de facto standard for representing shared data in a domain-independent way. According to the Semantic Web specifications, the domain-specific model underlying RDF data (data represented in RDF) can then be specified using the Web Ontology Language[9] (OWL). By relying on the combination of RDF and OWL, universAAL specifications already cover many of the features assumed for the model part of XForms. For example, the type system embedded in XForms Model is inherently supported in a more powerful way in universAAL because of using OWL-based ontologies. Therefore, the UI Framework of universAAL uses XForms specification just for the purpose of specifying a dialog package that consists of the equivalents of XForms controls; the data section of XForms is then replaced by RDF resources or OWL individuals originally used by universAAL applications.

The dialog package of universAAL embeds both the form controls and the form data in a single RDF resource of type “Form”, as illustrated by Figure 13.

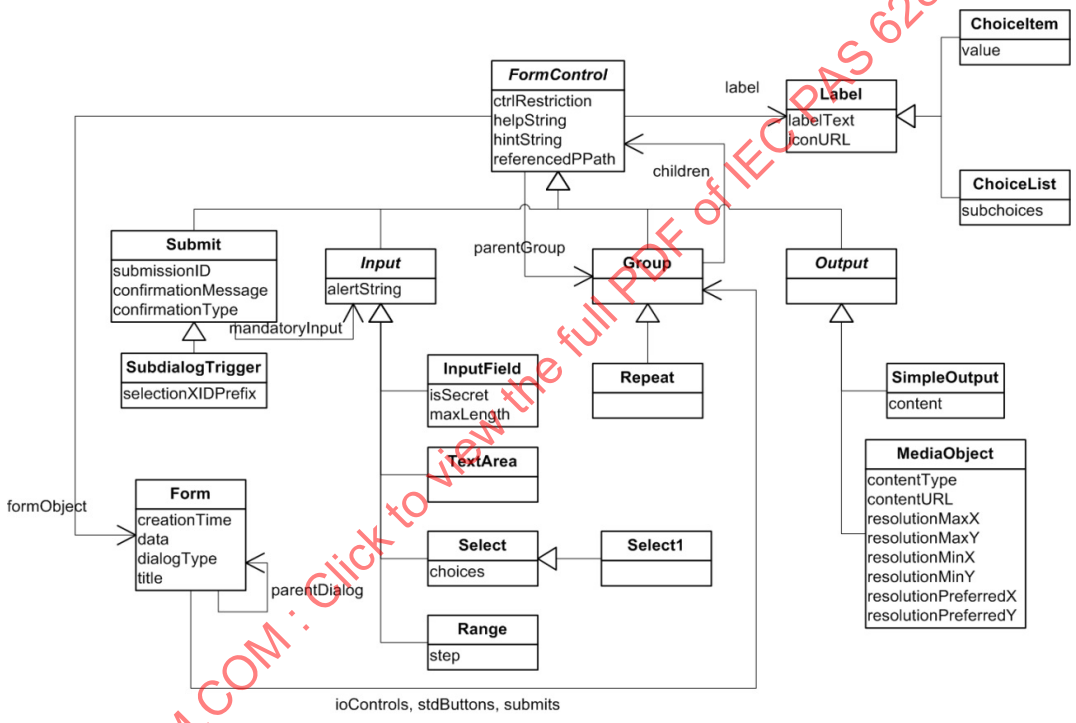



**Figure 13 – The dialog package based on the notion of a form**

Characteristics of this package are[10]:

- The dialog package is only inspired by XForms and does not provide any one-to-one implementation of the original set of form controls:
  - Originally, there is no hierarchical relationship between the form controls in XForms; the provided class hierarchy is the result of analyzing the nature of those controls.



7 http://www.w3.org/standards/semanticweb/

8 http://www.w3.org/RDF

9 http://www.w3.org/2004/OWL

10 Please refer to API Documentation available under http://depot.universaal.org/ for a complete documentation of this package. This API documentation exists in terms of Java API docs; the documentation of the dialog package can be accessed by navigating to the package org.universAAL.middleware.ui.rdf.

  - For some form controls in this package such as Sub-dialog Trigger and Media Object, there is no direct counterpart in the XForms original set of form controls.

- There are four types of dialogs in universAAL like Figure 14 shows:
  - System menu: the main menu assumed to be provided by a specific component of the UI framework, called the Dialog Manager, that provides an entry-level access to the available AAL services and applications.
  - Ordinary / standard dialog: to be used when the application wants to wait for a related input event upon which the application might decide whether to finish the interaction with the user or proceed to a next dialog.
  - Message: to be used when the application does not expect any input from the user as long as it can be sure the message will be delivered to the user and acknowledged by him or her.
  - Sub-dialog: to be used for constructing a hierarchy of running dialogs to ensure that control is returned to the parent dialog whenever a sub-dialog is finished.

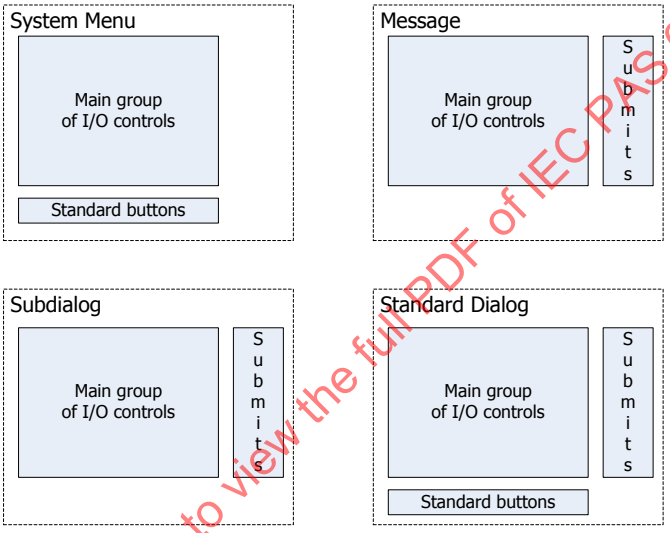



**Figure 14 – A possible graphical visualization of the mapping between dialog types and the predefined standard groups**

- Depending on the dialog type, a form in universAAL will consist of different combinations of the following predefined groups:
  - *submits*: "buttons" that finish the dialog intended by the current form should be added to this group;
  - *ioControls*: all other form controls, no matter if input or output controls, subgroups or even submits that trigger a sub-dialog should be added to this group;
  - *stdButtons*: this group is reserved for a dialog management solution that has access to all dialogs and may be willing to add standard buttons beyond the application logic to reflect a system-wide behaviour.

The reference implementation of the universAAL dialog package incorporates additionally the following features:

- The data part of the forms is hidden to the UI handlers so that they are relieved from error-checking when users provide input: UI handlers simply try to store user input provided in the context of a certain form control to that control element; if this attempt fails, the UI handler shall give some hint to the user using the alert message set by the application with the same semantic as defined by XForms.
- When a UI handler finishes a dialog, the data section of the form is automatically added to the response to be provided to the UI broker.

- Form controls can be generated by the dialog package automatically based on ontological knowledge associated with the data.

### 4.4 The Adaptation Concept

#### 4.4.1 Overview

The separation of the application and presentation layers based on introducing a brokerage mechanism in-between makes it possible to provide an efficient and effective approach to the accessibility and adaptivity challenges in the AAL domain. This approach is based on the division of the adaptation tasks between the three parties in a natural way:

- Applications may concentrate on adaptivity in control and in content composition, for example by sorting the application data to be presented to a user in a personalized and situation-ware way;
- UI handlers may concentrate on adaptation in rendering by taking the characteristics of the used channels into account; and
- The brokerage mechanism residing in-between may concentrate on the selection of the right UI handler for a given user in a given situation.

All the three layers can benefit from the universAAL framework for supporting adaptivity, both in terms of context-awareness and personalization, by subscribing for relevant contextual data and / or fetching such data. Figure 15 provides an overview of the corresponding set of components that comprise the universAAL framework for supporting adaptivity as an extension to the universAAL bus system (see footnote 4). Further discussion of this part of the universAAL platform is not in the scope of this specification.

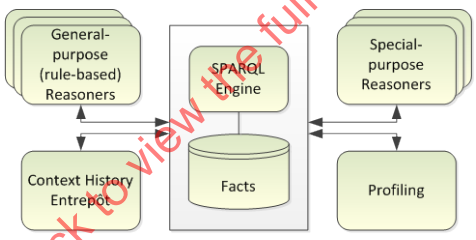



**Figure 15 – The universAAL framework for supporting adaptivity, which builds on top of the universAAL context and service buses (see footnote 4)**

#### 4.4.2 Responsibilities of Applications

Applications can prepare the content to be presented to the user in an adaptive way before adding it to a form object. Behaviour related to preparing the content in a personalized and situation-aware way is a choice that shall be made by the application developer and the UI framework cannot influence this behaviour largely. Here, the expectation is that the platform as a whole provides means for supporting adaptivity, quite similar to the universAAL framework outlined in Figure 15.

Applications shall contribute to the adaptivity of the UI framework as a whole. To enforce this, the framework requires that applications provide the following data items when making a UI request:

- Addressed User: the specific user whom the application wants to reach.
- The dialog priority: one of none, low, middle, high, or full; it will be ignored if at the time of receiving the UI Request no other dialog is running that involved the same user as the one addressed; otherwise, it will be used to determine whether the running dialog should be interrupted; if the running dialog shouldn't be interrupted, the request will be put in a queue sorted according to the dialog priority and time of arrival.
- The content language: a value of type *language* [6] as defined in the specification of XML Schema.

- The privacy level of the content: one of insensible, known_people_only, intimates_only, home_mates_only, or personal.

Applications shall also contribute to the rendering phase by providing alternative versions of any media objects used as content in the intended dialog. For this purpose, they shall rely on a specific component of the universAAL UI framework, known as the Resource Manager (see the specification of the RM in Section 4.5.4), by:

- referring in a generic way to a class of alternative media in dialog descriptions instead of referring to concrete resources; and
- providing the RM with several different versions of the media objects along with specific metadata, each for a specific runtime context.

### 4.4.3 Responsibilities of UI handlers

As components that utilize certain output channels for holding dialogs with human users, UI handlers shall adapt the modality- and layout-neutral description of dialogs to the capabilities of the devices that realize the utilized output channels. For performing such adaptation these rules shall be followed:

- The profiles of the devices that realize the output channels may be stored in the database of shared facts as indicated in Figure 15 (see in the same Figure also the component called Profiling); alternatively, the developer of a UI handler may rely on the API of the device driver used.
- The adaptation to the channel capabilities will necessitate transformations (e.g., text to speech) and layout-related decisions (e.g., the order of going through the dialog elements if a “one-dimensional” device such as a loudspeaker is being used for rendering the dialog). This specification makes no assumptions about the relationship of UI handlers to possible related transformation and rendering tools. From the perspective of the whole universAAL platform, it is recommended to decouple such auxiliary software and utilize their capabilities as shared services over the universAAL service bus (see footnote 4).
- The adaptation of media objects to the capabilities of the used devices needs the help of application developers or third parties. If the application developer does contribute and the dialog description contains references to the Resource Manager (see the recommendation to application developers in the previous section), the mandated UI handler will have to retrieve a best-match version of the referenced media object from the Resource Manager by providing appropriate parameters about the runtime context.

Similar to the case of applications, UI handlers shall contribute to the adaptivity of the UI framework as a whole. To enforce this, the framework requires that UI handlers shall provide their profiles to the brokerage layer when registering to that layer. The profile of a UI handler shall include the following information items:

- Supported channels: each channel shall be described by the following properties, which can be extracted from the profile of the device that realizes the channel and is utilized by the UI handler:
  - Channel type: input or output;
  - Location where the channel is available: will depend on the location of the corresponding device;
  - Interaction modality supported through the channel: currently, one of GUI, Speech, or Gesture;
  - Channel privacy level: private, public, or both; e.g., a headphone provides a private channel;
  - but the loudspeaker of a TV, a public channel, and the speaker of a phone provides both;
  - Modality-specific tuning capabilities: e.g., the volume range.
- Supported languages: a list of values of type *language* [6] as defined in the specification of XML Schema.

- Appropriateness for certain impairments: UI handlers might be specialists in interaction with certain types of users having certain types of impairments:

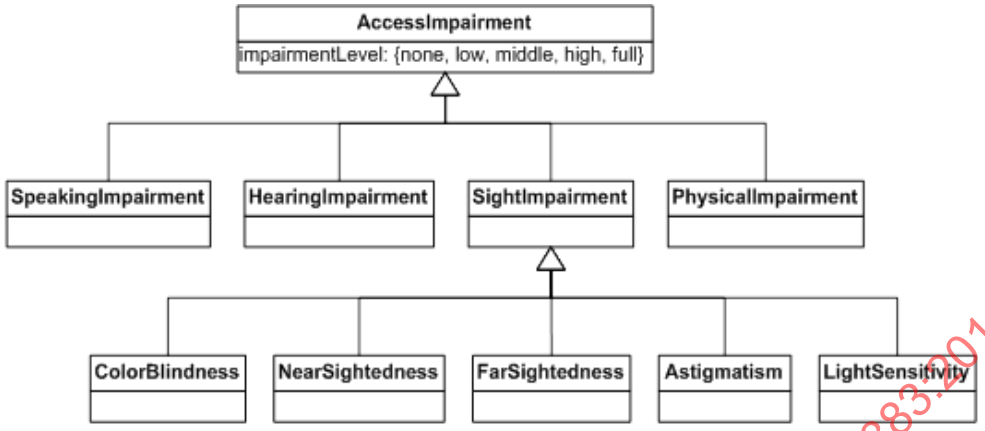



**Figure 16 – A model for describing access impairments**

### 4.4.4 Responsibilities on the brokerage layer

AAL spaces are open spaces; applications and UI handlers are "third-party" components that can be plugged into AAL spaces as desired by the users. The extent of adaptivity in the behaviour of these components is a function of the design decisions made by their developers. The level of adaptivity in these components will play a decisive role in determining their market shares.

In terms of the whole system, AAL spaces can still achieve a certain level of adaptivity even independently from the adaptivity level supported by the concrete set of applications and UI handlers running in the space. The prerequisite for this is the contribution of applications and UI handlers to a limited extent (4.4.2 and 4.4.3). Given this contribution, the following behaviour can be realized that is assumed to be essential for AAL spaces of future:

- enable natural interaction with exciting user experience by selecting a UI handler, which utilizes those I/O channels that are the best match for interaction with a certain user in a given situation;
- during the time a certain dialog is being held with the addressed user, instruct the UI handler to adapt itself to the changes in the context as far as the capabilities of the UI handler allow to cope with the changes; otherwise, transfer the remaining part of the dialog from that UI handler to a new one that is a better match for the changes in the user context. Examples for the effects that can be achieved are:
  a) adapt within the scope of modality-specific tuning capabilities of the used channels, e.g., perform a speech-based interaction a little bit louder because the situation with the background noise worsens;
  b) switch to another channel due to the change of the user location and achieve "follow me";
  c) switch to another channel with a different privacy level, e.g., when the personal content in the dialog should not be disclosed to someone that enters the same location as the interacting user;
  d) suspend the current dialog and start with another dialog with a higher priority;
  e) continue with a previously suspended dialog because the interrupting dialog finished or because the user instructs to do so;
  f) stop all interactions with the user because something else is attracting the user's attention.

For realizing the adaptation effects d) to f), there is need for a sort of dialog management, which is subject of discussion in Section 4.5.3. For selecting the most appropriate UI handler

and realizing the effects a) to c), the brokerage layer shall match the current state of a set of relevant parameters against the profile of the available UI handlers, each time that a UI request is received from the application layer. The corresponding set of parameters can be determined in the following way:

- use the content language indicated by the application for comparing it against the set of languages supported by the UI handlers available in the current AAL space;
- fetch from the user's context[11]: any impairments that the user addressed in the UI request might have and compare it against the appropriateness for access impairments claimed by the available UI handlers;
- fetch from the user's context: the current location of the user addressed in the UI request and use it for matching against the location of the output channels utilized by a given UI handler;
- fetch from the user's context: the modality recommended in the current situation for the user addressed in the UI request and match it against the modalities supported by the output channels utilized by a given UI handler [12];
- for the modality recommended, fetch from the user's context: the modality-specific tuning parameters preferred by the user addressed in the UI request and use it for matching against the modality-specific tuning capabilities of the channels to use;
- fetch from the user's context: the currently highest privacy level that a dialog is allowed to possess in order to still be held using public channels and compare it with the privacy level provided by the application in order to determine the privacy level required from the output channel to use [13].

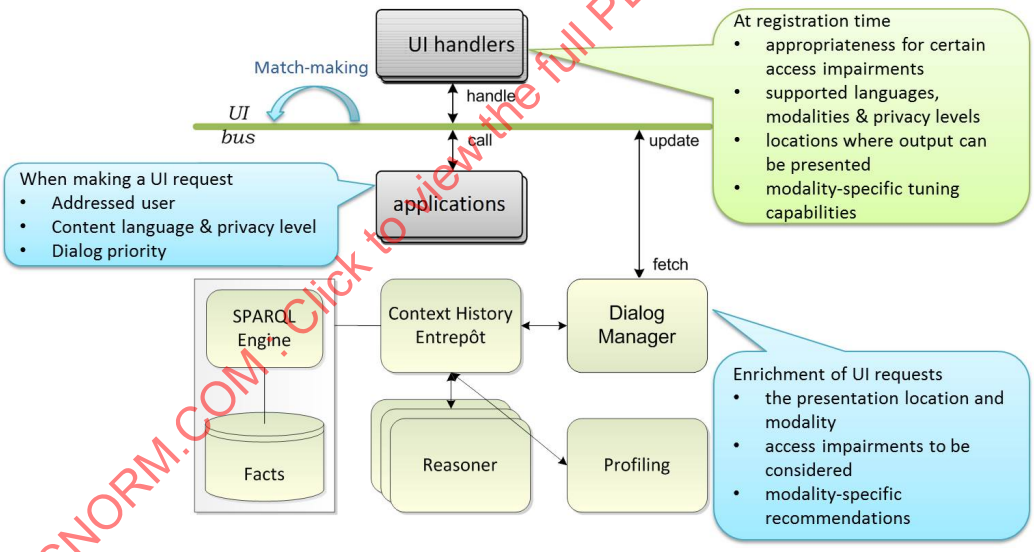



**Figure 17 – Summary of the adaptation parameters**

NOTE When realizing the matchmaking procedure, the brokerage layer should fetch in a first operation all of the parameters, then go through all of the profiles registered by the available UI handlers and match the fetched parameters against those profiles, while ranking them. In the end of this process, the UI handler whose profile ranked the best will be selected. See also the matchmaking rules in Section 4.5.2.



---

11 Assumes that the user model includes the corresponding information. In universAAL this model exists in terms of an ontology, and all info describing user context is assumed to be available in the "facts database", which is part of the universAAL framework for supporting adaptivity; see also footnote 15.

12 In universAAL, a rule determines this value. One of the general-purpose reasoners in the universAAL framework for supporting adaptivity (more specifically, the so-called Situation Reasoner that works based on SPARQL) monitors the changes in the context, and based on those changes, re-evaluates the affected rules. The evaluation of these rules might lead to the construction of a new fact that is then fed into the "fact database". This always-running background process ensures that a simple fetch operation is enough for, say, retrieving the most up-to-date recommendation for the modality to use in the interaction with a given user.

13 In universAAL, a rule determines this value as in case of the recommended modality.

## 4.5 Provisions of the UI Framework

### 4.5.1 Introduction

There are three components as integral parts of the universAAL UI framework:

a) The UI Bus, which provides a message brokerage mechanism that facilitates the independence of the applications and UI handlers from each other.

b) The Dialog Manager, which assists the UI Bus in the management of dialogs in order to achieve some of the desired adaptation effects listed as examples in Section 4.4.4.

c) The Resource Manager already mentioned in Sections 4.4.2 and 4.4.3.

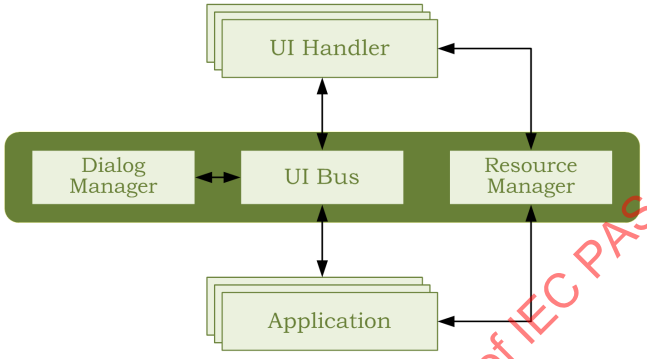



**Figure 18 – The components comprising the universAAL UI framework**



### 4.5.2 The UI Bus and its brokerage protocols

In general, messaging brokers can be abstracted as virtual communication buses, or short *buses*. This is also the abstraction used by the universAAL platform; therefore, the broker in the universAAL UI framework is called the *UI Bus*, which is mandated to facilitate the interaction between applications and UI handlers.

A bus may be realized in a centralized way so that only one single instance of the bus exists in a whole system; if the UI Bus is realized in that way, there would be a 1:1 relationship between the term UI Bus and the singleton object that realizes the bus. Alternatively, a bus can be realized in a distributed way with one instance per runtime environment so that components that want to connect to the bus can do so by accessing the local instance of the bus and use its API natively. In that case, the different instances of the same bus running in a distributed way within different runtime environments shall find each other and cooperate in a way that the distribution and possible heterogeneity of the different runtime environments can be overcome. If so, a message posted by a component to the bus instance that exists in the same runtime environment may reach another component running in another runtime environment, when necessary. Through such cooperation between the different instances of the same bus, each instance will be perceived as the footprint of a single global bus by the components locally attached to that instance. If the UI Bus is realized in this way, then the term UI Bus would refer to the logically global bus that emerges through the cooperation between the different instances of the bus distributed among different runtime environments[14].

Buses can operate either event-based or call-based. Event-based buses support the communication pattern known as publish-subscribe, whereas call-based buses realize the request-response communication pattern. The UI Bus in the universAAL UI framework operates call-based and consequently supports the request-response communication pattern. As a result, the main messages posted to the UI Bus or forwarded by the UI Bus to its members are called *UI Request* and *UI Response*, respectively.

---

14 The reference implementation of the UI Bus in the universAAL platform is based on the second paradigm as part of a distributed middleware approach that consists of two other buses, namely the context and service buses, as indicated in footnote 4.

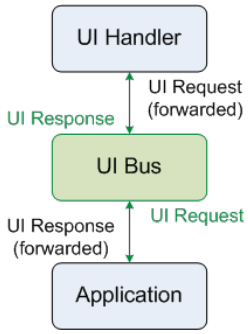



**Figure 19 – The main messages exchanged on the UI Bus**

From the viewpoint of an application, a UI Request consists of a form object as described in Section 4.3. and the set of parameters described in Section 4.4.2. The resulting object model can be summarized as follows:

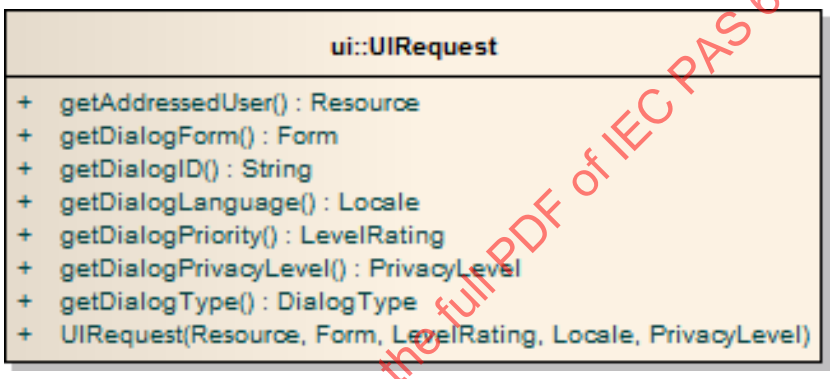



**Figure 20 – The notion of a UI request as constructed by applications**

When an application posts a UI Request to the UI Bus, the latter asks the Dialog Manager to (1) make a decision about handling the request in the current situation (4.5.3), and – if appropriate to be handled immediately – (2) augment the request with up-to-date information from the user context according to the logic described in Section 4.4.4. As a result, the UI Request will be enriched by a set of parameters, such as user's possible access impairment, recommended presentation modality for the addressed user in her/his current situation, alternative presentation modality recommendation, presentation location (where the user finds himself currently), presentation privacy, and modality-specific user preferences. The enriched UI Request is then matched against the profiles of registered UI handlers (see the enumeration in Section 4.4.3 for the content of these profiles) in order to choose the most appropriate UI handler.



The following rules are used during matchmaking in order to assign scores to the candidate UI handlers before choosing the one with the highest score:

- Omit candidates not able to present a dialog in the required presentation location.
- If the usage of private channels is required by the enriched UI Request, omit candidates not offering this capability.
- If the last UI handler that was interacting with the user is among the remaining candidates, take that one as the best match.
- If accessibility support is mandatory, omit candidates not appropriate; otherwise assign higher scores to candidates offering appropriate accessibility support.
- Omit candidates that support neither the recommended presentation modality nor the alternative modality; then, assign higher scores to candidates that support the recommended presentation modality rather than supporting the alternative one.

- Assign higher scores to candidates that provide a better match for the modality-specific user preferences.
- At any time after omitting not appropriate UI handlers from the list of candidates, if the list of remaining candidates becomes empty, then try to find a compromise for the conflicting situation; e.g., if the enriched UI Request indicates that only private channels shall be used but no UI handler offers this capability in user's location, then the user shall be asked if she/he would change to a location with available private channels, or the information should be presented in the same location using public channels, or the interaction is postponed to a later time. By default – e.g., when there is no way to ask the user for resolving the conflicting situation – the UI Bus will ask the Dialog Manager to preserve the dialog for a later time.

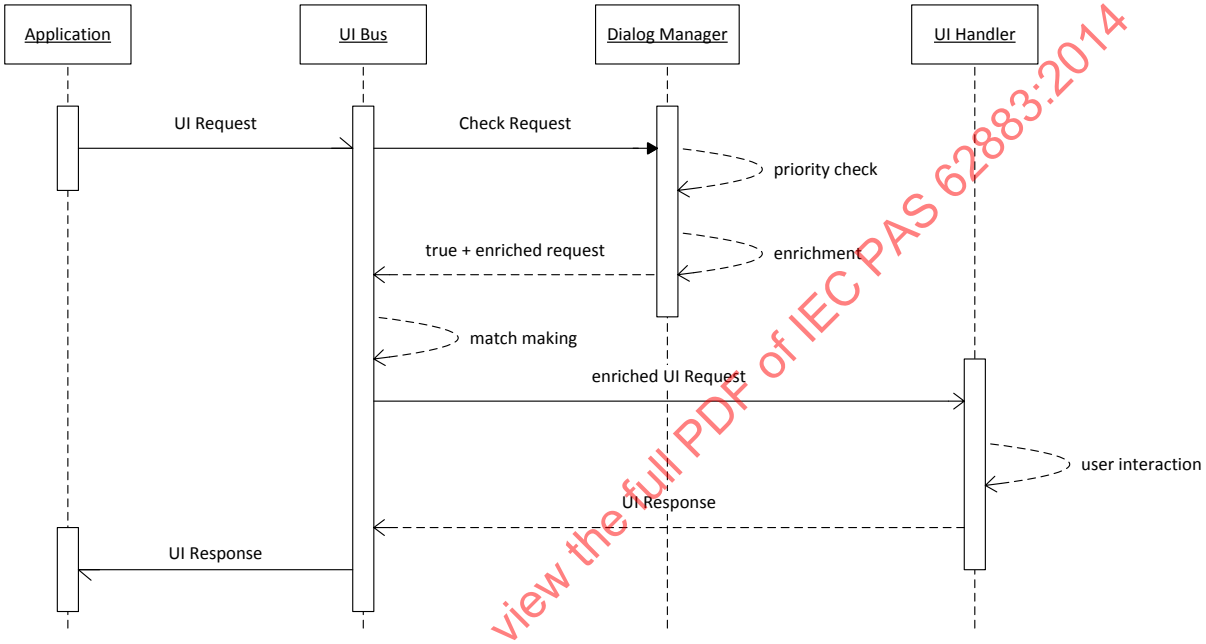



**Figure 21 – Overview of the sequence of actions when the priority check is positive**



If any of the above parameters from the enriched UI Request changes during the time after mandating a UI handler to perform the corresponding dialog with the user and before the UI handler reports that the dialog has been finished, the Dialog Manager resends the enriched UI Request with its updates to the UI Bus and requests to redo the matchmaking. Examples of such changes are:

- If during interaction with a UI handler the user changes the location from one room to another, the Dialog Manager will have to update the presentation location in the enriched UI Request.
- If during interaction with a UI handler a third person who is in conflict with the privacy level of the presented content, enters the presentation location, the Dialog Manager will have to update the required channel privacy.
- If, according to some user preference, the loudness of the speakers used for interacting with the user should change depending on the loudness of the background noise, and during speech-based interaction with a UI handler, the background noise changes significantly, the Dialog Manager will have to update the modality-specific parameter "volume".

After receiving the updated UI Request, the UI Bus will redo the matchmaking with the profiles of available UI handlers; if as result the newly matched UI handler is the same as the UI handler matched previously, this UI handler will be notified about the change of the enriched UI Request in order to adapt its behaviour to the changes, e.g. by transferring the rest of the interaction to more appropriate channels accessible to it (channels in the new location of the user or private channels in the same location) or by adapting the modality-specific rendering

parameters (e.g., speech output becomes louder). If, however, the new UI handler is different from the old one, the old UI handler will be asked to cut the dialog and return the form data with all intermediate user input gathered so far, then the enriched UI Request updated by the Dialog Manager will be re-updated by the UI Bus in order to include the changes in the form data, and finally the new UI handler will be mandated to handle the dialog with the user.

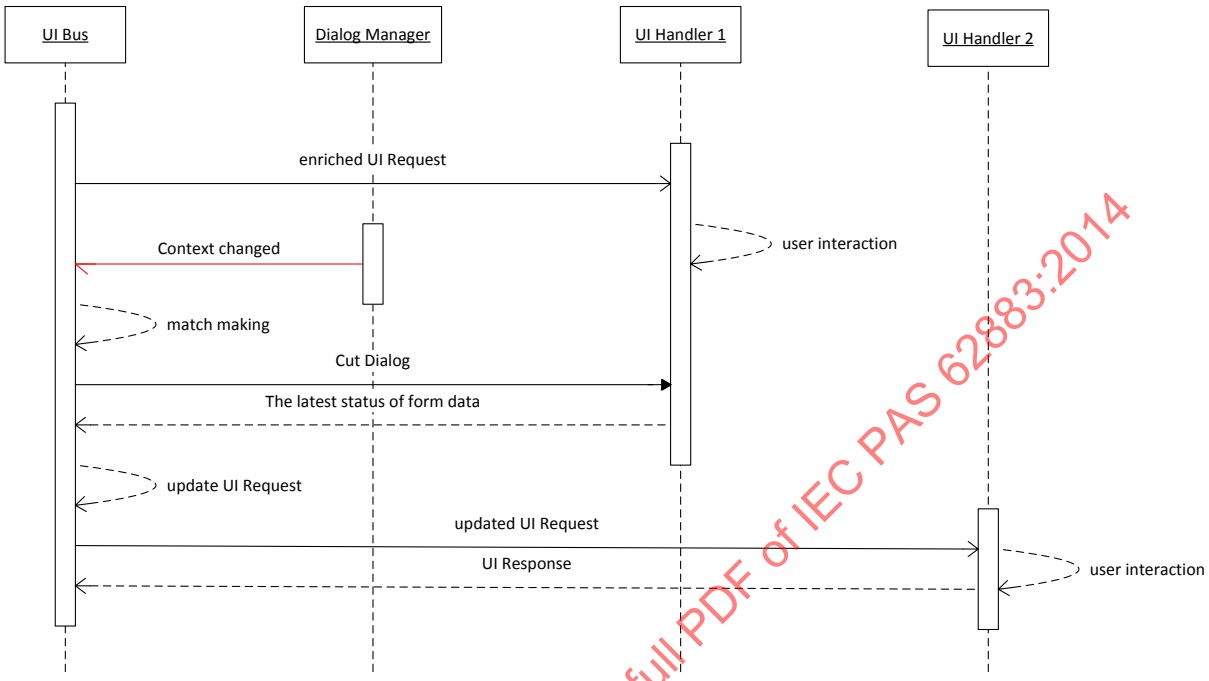



**Figure 22 – The case of switching to a new UI handler when handling changes in the context**



Beside this main protocol for forwarding UI Requests from applications to UI handlers, the UI Bus supports the following additional protocols:

1) Finishing a dialog

When a UI handler finishes a dialog with the user, it constructs a UI Response by providing the following set of data and hands it over to the UI Bus:

- The user who provided the response.
- The location where the input was provided.
- The ID of the dialog finished.
- The final state of the form data as updated during the dialog with the user.
- The ID of the parent dialog, if the finished dialog was a sub-dialog of another dialog running previously (see the discussion of sub-dialogs further below).
- The ID of the “Submit” object that caused the dialog to finish (4.3).
- A Flag indicating if the above “Submit” object actually does not finish the dialog but simply suspends it as a result of activating either a “SubdialogTrigger” (see both the description of the dialog package in Section 4.2 and the discussion of sub-dialogs further below) or a “standard button” (see both the description of the dialog package in Section 4.3 and the discussion of the standard buttons in Section 4.5.3)

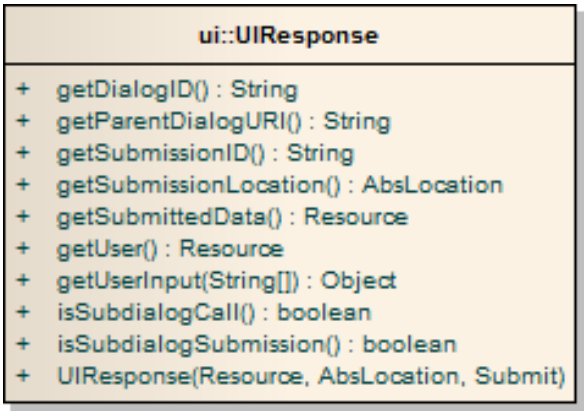



**Figure 23- The notion of a UI response as constructed by UI handlers**

Upon receiving a UI Response, the UI Bus forwards it to the application that had made the original UI Request and notifies the Dialog Manager that a dialog with a certain ID has finished or been suspended. See section 4.5.3 for the reaction of the Dialog Manager to this notification.

2) Aborting a dialog

There are three ways for aborting a dialog: (1) the user may decide to abort a running dialog using “Submit” objects provided by the application itself; in that case, the abort action actually falls into the normal category of “finishing a dialog” as discussed in the previous bullet; (2) the user may decide to abort a suspended dialog using a standard dialog designed by the Dialog Manager (4.5.3); in that case, the Dialog Manager will ask the UI Bus to inform the corresponding application about this action of the user; (3) the application itself decides to cancel its UI Request (e.g., because a deadline for the expected user input was missed); in that case, the application shall also provide a human readable explanation for this action and wait until the UI Bus either confirms the abort or informs that the dialog has finished normally.

Upon receiving such an abort request, the UI Bus checks if the corresponding dialog is running; if yes, the request to abort will be forwarded to the UI handler in charge of handling the dialog. The UI handler is expected to communicate this situation with the user while referring to the reasons provided by the application. The result of the user decision will then be communicated both to the application and to the Dialog Manager. If, however, the application is asking to abort a dialog that had been suspended for whatever reason and hence is not running, then only the Dialog Manager is informed by the UI Bus to remove the dialog from its list of suspended dialogs and not reschedule it for later resumption.

3) Resumption of a previously suspended dialog

A dialog can be suspended due to four reasons (in that case, the Dialog Manager will put the dialog in a user-specific queue of dialogs waiting for resumption): (1) at the time when the application made its UI Request, the addressed user was already involved in another running dialog with the same or higher priority; (2) a running dialog might be suspended because a dialog with a higher priority has to be pushed into the foreground; in that case, the UI handler in charge of the running dialog will be asked to cut the dialog and return the form data with all intermediate user input gathered so far in order to forward it to the Dialog Manager for suspension; (3) when the user activates a “SubdialogTrigger” in a running dialog, the latter will be suspended; (4) when a user activates a “standard button” in order to switch to a standard dialog provided by the Dialog Manager; such standard buttons are added to application dialogs by the Dialog Manager during the enrichment of UI Requests; activation of these buttons leads to the suspension of the running dialog.

Resumption of a previously suspended dialog may occur automatically after a dialog with no parent dialog finishes. In that case, the notification sent by the UI Bus to the Dialog Manager causes that the latter checks the user-specific queue of dialogs waiting for resumption and picks the dialog with the highest priority for resumption; in case that there are several dialogs

with this priority, the oldest one from among them will be selected. If the queue is empty, then the default standard dialog (the system “main menu”) will be selected. At this time, the selected dialog (UI Request) will be updated with data from the user context and the UI Bus will be asked to start with a new matchmaking process for mandating an appropriate UI handler to resume with that UI Request.

However, if the finished dialog was a sub-dialog, the Dialog Manager won’t execute the above procedure for resumption; rather, the system expects that the application that had made the corresponding UI Request will process the input data provided by the user and incorporates any resulted changes in the data associated with the parent dialog and ask the UI Bus to resume with the parent dialog. At this stage, the UI Bus restarts the whole process with the enrichment of the UI Request by the Dialog Manager with up-to-date data from user context, matchmaking, and mandating the most appropriate UI handler (usually the same UI handler that just finished the sub-dialog).

The need to resume a suspended dialog may also arise when in the context of a standard dialog provided by the Dialog Manager, the user explicitly chooses a waiting dialog for resumption. Such an action finishes the corresponding standard dialog so that the Dialog Manager will then resume exactly the dialog chosen by the user instead of following its own logic for the priority-based selection of the next dialog.

4) Handling sub-dialogs

Sub-dialogs are usually used for structuring complex dialogs in a neater way. In the usual case of GUI-based interaction, a sub-dialog is usually rendered in a pop-up window when some action of the user activates the sub-dialog. Then, usually the user cannot return to the parent dialog before the sub-dialog is finished. A reason for this behaviour is that the user input in the context of the sub-dialog may affect the form data in the parent dialog so that the parent dialog may need to be re-rendered. A typical example for such a situation is given, when the parent dialog includes only a summary of more complex data related to the current dialog and hence has to embed a “SubdialogTrigger” (as it is called in the dialog package of the universAAL UI framework described in Section 4.3) in the current form in order to provide the user with the possibility to view all the related details; any changes to these details can usually be made to the detail data only in the context of such sub-dialogs.

When in the context of a running dialog the user activates a “SubdialogTrigger”, the UI handler shall construct a UI Response as if the current dialog has finished, and set in this response a flag indicating that the submission has been caused by a SubdialogTrigger. The standard “submission ID” in the response will then be the ID of the concrete “SubdialogTrigger”. Then, the UI handler should assume that most probably it will be mandated by the UI Bus to perform the sub-dialog that the application is assumed to send in immediate reaction to such UI Response. Therefore, it should use some interaction “trick” until the situation is clarified. When the application submits a UI Request that is in reaction to such a UI Response for activating a sub-dialog, it has to include the ID of the parent dialog in the corresponding UI Request.

When a sub-dialog finishes, the UI handler shall construct a UI Response in which the ID of the parent dialog is returned back to the application so that the latter can uniquely identify which original dialog is expected to be resumed.

Figure 24 to Figure 26 summarize the above discussion in terms of APIs.

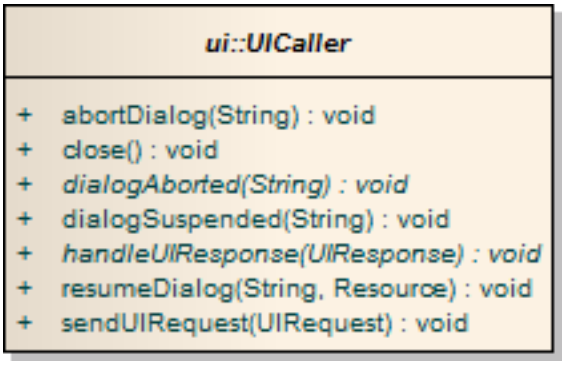



**Figure 24 – The abstract class to be extended by applications that want to send UI requests to the UI bus**

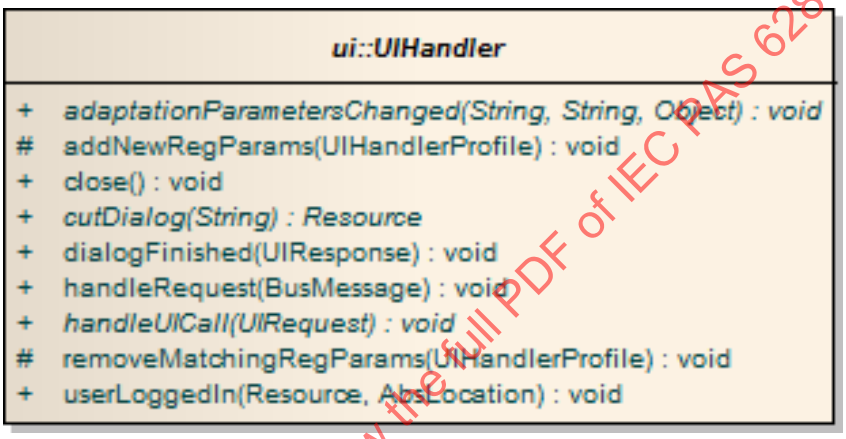



**Figure 25 – The abstract class to be extended by UI handlers that accept UI requests forwarded by the UI bus for rendering**

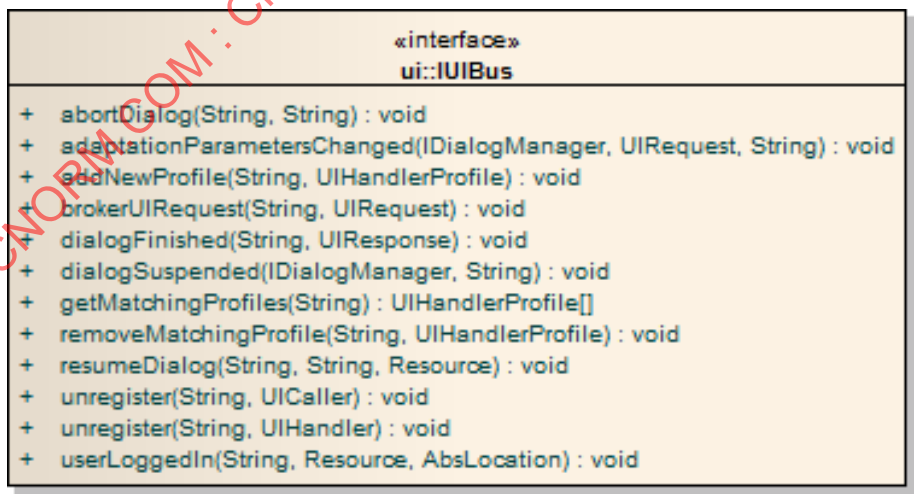



**Figure 26 – The interface of the UI Bus**

### 4.5.3 The dialog manager and its role in assisting the UI Bus

The Dialog Manager (DM) plays multiple roles in the universAAL UI framework for AAL spaces. It (1) represents the whole system by providing different flavours of system menus and standard dialogs, (2) assists the UI Bus by being responsible for the incorporation of user context in analyzing the interaction situations, (3) assists the UI Bus also by providing a

persistent mechanism for user-specific dialog management, (4) handles user instructions to the AAL space as a whole, (5) provides a mechanism for users to edit their UI related preferences as a specific standard dialog, and (6) is finally responsible for offering a unified view of all user services available in the AAL space as an alternative to system menus and a method to search for specific services. Due to its roles (1), (4), (5), and (6), it is logically also playing the role of an application that creates own UI Requests and sends them to the UI Bus for being brokered to an appropriate UI handler.

Many of the above features of the Dialog Manager have been already specified in the context of previous sections in this document. Therefore, this section only provides complementary information needed to specify the tasks of the Dialog Manager in more details:

1) Dialog Management

The DM manages parallel dialogs for several users based on priority queues of published dialogs and takes care that only one dialog is presented to the user at each point in time based on the priority of the UI requests received from the application layer. Before brokering UI requests to UI handlers, the UI Bus asks the DM to check if the UI request should be handled immediately or wait until a later time; if the result of this check is negative, the corresponding UI request will be queued and the UI framework does not undertake any additional action. Only if the result of the check is positive, the enrichment of the UI request with relevant up-to-date data from the user's context will be done immediately; otherwise, this step will be done at the time of resumption (see also the previous section).

One of the important aspects of the enrichment is the addition of "standard buttons" to the form object contained in UI requests. They make it possible for the user to view and manipulate the queue of waiting UI requests or switch to the "system main menu" (more on that further below). An important remark about the "system main menu" is that it is the first dialog presented to users as soon as they take the initiative to start interacting with an AAL space by accessing a desired UI handler. In such cases, the UI handler will inform the UI Bus that, in the absence of any history for a specific user, she/he has started actively to use it for interacting with the AAL space. The reaction of the UI Bus will be to ask the DM for the default dialog, which is the "system main menu" as assumed by the DM.

In addition to the user-specific queue of waiting dialogs, the DM remembers the currently running dialog for each user separately. This way, it can handle all the cases of completion, aborting, suspending and resuming dialogs correctly and in accordance to the protocols described in the previous section.

2) Standard Dialogs

Recurring system-wide and application-independent dialogs and dialogs related to the act of managing dialogs are called standard dialogs in the terminology of the universAAL UI framework for AAL spaces. The "standard buttons" – referred to previously several times – are expected to provide access to these kinds of dialogs.

Currently, the following are the only system-wide dialogs specified in this framework:

- The "system main menu" that provides for navigation through a specific organization of user services available in the AAL space based on user- and language-specific configuration files.
- A dialog for searching among all the user services available in the AAL space that can be accessed by a specific user in a free form, beyond the "system main menu", which is a pre-configured organization of them that might not be comprehensive enough.

  Searching for services usually occurs in the context of a need; hence, such a free search might be used for directly articulating a need, such as "turn on the light on the wall behind me!" This kind of "search" can actually be classified as an instruction to the AAL space, which is usually equivalent to a shortcut to some available service that otherwise would be found by navigating in menus. As a result, the DM is also in charge of resolving such instructions as far as possible.

- A dialog for viewing and manipulating UI-specific user preferences

In addition, currently supported dialogs related to the second category of standard dialogs, namely the act of managing dialogs, include two similar but separated dialogs for browsing, resumption or aborting UI requests that are waiting in the user-specific queue: One of them is specific to pending “messages” and the second one for all the other pending dialogs (see the list of dialog types in Section 4.3).

### 4.5.4 The Resource Manager

The Resource Manager (RM) is mainly responsible for managing application-specific media objects as resources to be used by UI Handlers when rendering UI requests. Applications may refer to such resources in their UI requests by using URIs; this “presentation URI” is mapped by the RM to alternative concrete URIs that can be accessed directly by the UI Handler. Both application vendors and third parties can then provide the RM with several different versions of the media objects, each for a specific runtime context. The RM stores the resources together with their URIs and relevant metadata persistently.

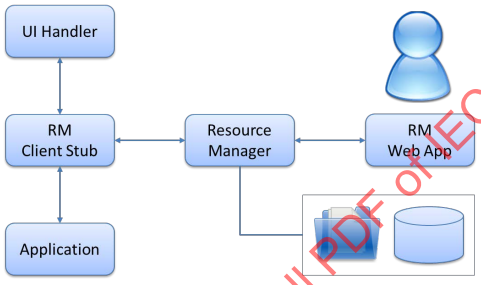



**Figure 27 – Access to the resources managed by the RM**

Application developers are expected to use the RM the following way:

a) In UI Requests, refer to media objects by a “presentation URI”.

b) For each “presentation URI” provide a set of alternative concrete resources, each of which to be described by the following set of metadata:
   - Mime type and related parameters (e.g., for images: resolution, colour, etc.).
   - Locale and related parameters.
   - Possibly a set of keywords.
   - The initial set of media objects and their metadata should be part of the application package; the application should forward them to the resource manager as soon as it starts to run. The administrator of the RM may extend the set of alternatives.

UI Handlers are expected to use the RM in the following way:

a) Ask the resource manager for substituting all “presentation URIs” found in the UI request at hand with the URIs of concrete resources by sending to the RM the UI request and a set of parameters describing the rendering context, such as the currently used modality and language, as well as preferences with regard to resource metadata using value restrictions (e.g., “if set” / “one of” / “in range”) and / or white, black or wish lists (wanted / unwanted / wished). In response, the RM will return the transformed UI request, in which all presentation URIs have been substituted from the URIs of concrete resources that are best matches for the given rendering context and value restrictions.

b) When rendering the transformed UI request from the previous step, the actual resources may be retrieved from the RM using the concrete URI found in the adapted UI request.