Funded by the Horizon 2020 Framework Programme of the European Union

Combined D2.1&D2.3
Interim report on adaptive personalized user interface component &
Interim report on adaptive personalized content component

PROJECT DOCUMENTATION SHEET	
Project Acronym	Easy Reading
Project Full Title	Easy Reading
Grant Agreement	780529
Call Identifier	H2020-ICT-2016-2017
Topic	ICT-23-2017
Funding Scheme	RIA (Research and Innovation action)
Project Duration	30 months (January 2018 – Juni 2020)
Project Officer	Michael Busch, European Commission Directorate-General for Communications Networks, Content and Technology, Unit &-3, L-2557 Luxembourg, +352.4301.38082
Coordinator	Universität Linz (JKU), Austria
Consortium partners	Kompetenznetzwerk (KI-I), Austria Technische Universität Dortmund (TUDO), Germany In der Gemeinde Leben gGmbH (IGL), Germany FUNKA Nu AB (FUNKA), Sweden Texthelp Ltd (TEXTHELP), United Kingdom VÄSTRA GÖTALANDS LÄNS LANDSTING (DART), Sweden GEIE ERCIM (ERCIM), France AHTENA I.C.T. Ltd (ATH), Israel
Website	www.easyreading.eu

DELIVERABLE DOCUMENTATION SHEET	
Number	D2.1, D2.3
Title	Combined report: Interim report on adaptive personalized user interface component & Interim report on adaptive personalized content component
Related WP	WP 4
Related Task	Task 2.1: Interface concept development, Task 2.2 Accessible information presentation
Lead Beneficiary	JKU
Author(s)	Peter Heumader
Contributor(s)	
Reviewers	Shadi Abou-Zahra
Nature	Report
Dissemination level	Public
Due Date	31.12.2018
Submission date	20.12.2018
Status	final

QUALITY CONTROL ASSESSMENT SHEET			
Issue	Date	Comment	Author
1	19.12.2018	General Review	Shadi Abou-Zahra

DISCLAIMER

The opinion stated in this report reflects the opinion of the authors and not the opinion of the European Commission.

The commercial use of any information contained in this document may require a license from the owner of that information. All Easy Reading consortium members are also committed to publish accurate and up to date information and take the greatest care to do so. However, the Easy Reading consortium members cannot accept liability for any inaccuracies or omissions nor do they accept liability for any direct, indirect, special, consequential or other losses or damages of any kind arising out of the use of this information.

ACKNOWLEDGEMENT

This document is a deliverable of the Easy Reading project, which has received funding from the European Union's Horizon 2020 Programme for Information and Communication Technologies under Grant Agreement (GA) Nb #780529.

Executive Summary

The present document is a the combined version of deliverable D2.1Interim report on adaptive personalized user interface component and D2.3 Interim report on adaptive personalized content component of the Easy Reading project which is funded by the European Union's Horizon 2020 Programme und Grant Agreement #780529.

This document gives an overview on how user interfaces within the EasyReading framework and the help provided by it will adapt to the individual user's needs. The document starts with a brief introduction on adaptive/adaptable user interfaces and how they can used to improve the accessibility of web-content for people with cognitive disabilities. It describes the workflow of the EasyReading framework and how the adaptivity of the framework is realized. This is underlined with examples of components currently implemented within the framework. The document ends with an overview of the adaptation strategies within the framework and an outlook on the features to come in the next year.

Table of Content

Disclaimer

This is the **combined** interim report on the **adaptive personalized user interface component** and the interim report on **adaptive personalized content component**. As the components are very dependent on each other, individual reports would either result in duplicate information in each report or cross-links between the two deliverables, making them more difficult to read. Therefore, this combined report was created.

Introduction

The concept of user interfaces that would change according to the requirements, skills, environment, situation, or other criteria has been around for a long time. In general, these concepts can be categorized in adaptive user interfaces and adaptable user interfaces.

- Adaptive User Interfaces [1]: These systems change their structure, functionalities, and content for the individual user in real time. This is achieved by monitoring the user status, the system state, and the current situation that the user is facing. By using an adaption strategy (mostly rule based), the user interface is changed at run time.
- Adaptable User Interfaces [2]: This user interfaces are highly adjustable in terms of presentation of information, display of user interface and its components or user interaction/input concepts. The settings are usually stored in a user profile and the user is able to adjust those settings in advance, usually in a settings dialog. During runtime, in contrary to the adaptive user interfaces, these settings do not change.

User interface adaptations show great potential towards usability and accessibility of systems. User tracking with state of the art sensors could give estimations about the current user's status, and could trigger adequate system reactions based on that. [6]

According to Brusilovsky [3], user interface adaptations for web applications can take place in three areas:

- Selection of content: The content that is presented to the user within the user interface
- Presentation of information: This deals with the visual presentation of all information within the user interface. It includes colours, font sizes, layout, etc.
- Navigational concepts: Describes how users navigate through the interface in order to gather the desired information or functionality provided by the webpage.

Adaptive user interfaces that aim at improving accessibility will have to take the great diversity in perceptual, motor, and physical abilities into account. For contextual conditions, the webpage must be adapted in content, presentation and navigation [6].

However, adaptations on a webpage that would improve accessibility for a user might have some unwanted side effects. For example, increasing the font size to address the vision impairment of a person might result in longer text passages and the need to scroll, which requires increased attention and memory demands for the user. Therefore, providing extensive adaptability is a highly complex task, as side effects and conflicts are difficult to locate.

Methods of User Interface Adaptations

According to Laive[4], methods for user interface adaptations can be assigned to the following categories:

- Adaptable/Manual: the user manages the process and performs all actions;
- Adaptable with system support/User Selection: the user dominates the adaptation process and the system supports it;
- Adaptive with user control/ User Approval: the system dominates the adaptation process under the supervision of the user. The system initiates the action and notifies the user about the alternative that he/she has to choose;
- Adaptive/Fully adaptive: the whole process is managed by the system, which decides and implements the action based on the preferential model and the main uses.

Adaptable/manual systems have the advantage that the user is in total control of the user interface. The obvious drawback is that the user needs to understand the configuration mechanisms that are offered by the system. For simple systems, the configuration dialogs might be small and simple. However, a system that allows many features to be customized for the individual user will result in a very complex settings dialog, as each feature needs some sort of configuration form. This is especially a problem for people with some forms of cognitive disabilities, as they are overwhelmed by the complexity and amount of choices offered by the settings dialog. [5]

Adaptable with system support is more user friendly then pure adaptable systems. Here the system provides support when configuring and personalizing the user interface to suit the user's needs. An example for such support is a wizard that guides users through the configuration process. These wizards are usually stepwise and linear, giving a more user friendly way of configuring the system.

Adaptive with user control/ User Approval are systems where the user interface automatically adapts to the user's needs if the user approves the adaption. This is adaption is based on monitoring the user status, the system state, and the current situation that the user is facing in real time. When the system detects a change that would result a change in the user interface, it would notify the user. Usually this is done by a simple dialog informing the user that the system has an optimization, and how this optimization would look like. Most important, the system asks the user for permission to perform the optimization. If the user declines, no changes to the user interface are made. For many people with cognitive disabilities this might be a good choice, because they are informed that something is going to happen and it is also explained what is going to happen. However, the dialog also might distract the user from the actual task that he/she is facing, especially when the system is too reactive and would apply frequent changes.

Adaptive/Fully adaptive systems in contrary to the adaptive systems with user control do not ask for permission when the system is changing the user interface. This results in a more invasive approach where the system takes care of all the configuration without any direct interaction of the user that would allow to change the current state of the user interface. As there is literally no configuration needed – assuming that a valid user model/profile already exists – these systems require the least cognitive load to configure a personalized user interface. This approach requires, however, that the user model/profile is complete and correct and covers all aspects of the parameters to be adapted. If this is not the case then features of importance might be excluded or unnecessary features are added, leading to unwanted information within the user interface.

Benefits for People with Cognitive Disabilities

As the abilities and preferences of people with cognitive disabilities are different for each individual person a "one size fits all" approach is not appropriate. People with cognitive disabilities need highly

individual support when it comes to understanding content and operating user interfaces. In addition, human computer interaction paradigms, presentation of content or user interface concepts might be helpful for an individual user with cognitive disabilities but rejected by other people with cognitive disabilities [7].

An example of such a contradictory situation might be a video explaining a website before use. This can be superfluous or even annoying for mainstream users but very helpful if not necessary for the target group. The same applies to the so called “mouse-over function”, that might be helpful to understand the meaning or function of a word or link, or the use of specific colours to explain contextual relations. This is highly beneficial when used to enrich the given information and eases perception (e.g. red (“be careful”) / green (“proceed" / "clear”) [7].

Adaptive and adaptable user interfaces are therefore very beneficial for people with cognitive disabilities as they can be adjusted for the individual need – either automated or manually.

Workflow within the Easy Reading Framework

User will interact with the EasyReading framework with clients that are implemented as browser plugins for Firefox or Chrome or with apps for iOS and Android. Once logged in, the framework will fetch their personalized user interface with personalized functions from the cloud and inject it into the webpage. As the user interface is part of the webpage, it is based built with HTML, JavaScript, and CSS. (For a more detailed description on that please see Deliverable 6-2 Initial Architecture Report).

Once the user interface is injected, the user is able to create a request with one of the widgets that are part of the user interface (See figure 1).

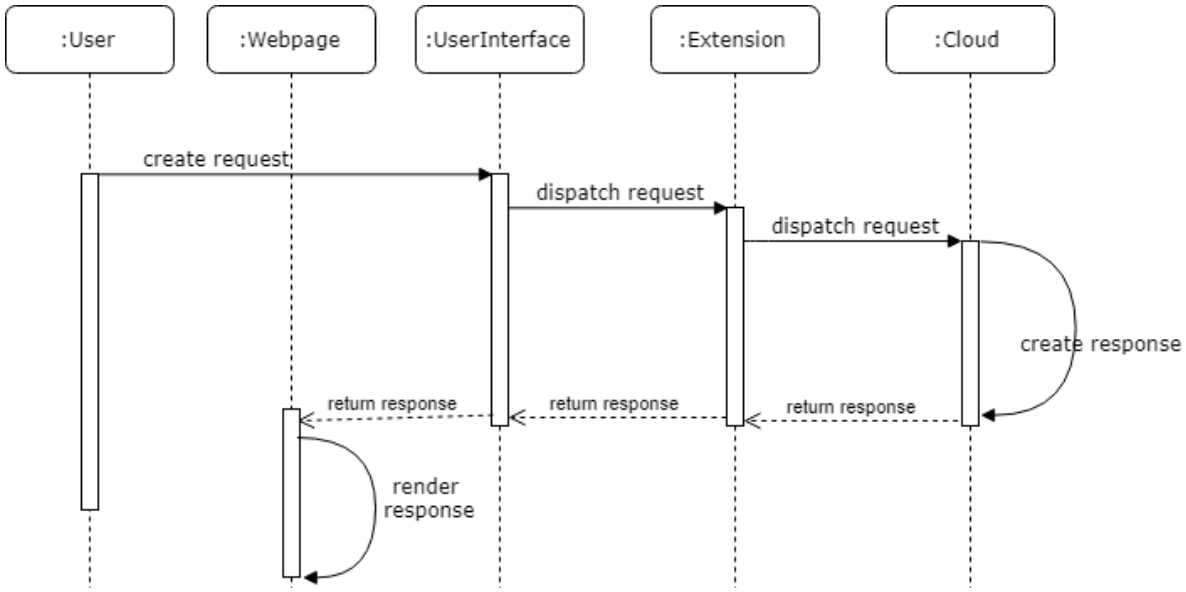

Figure 1: Interaction diagram of the workflow

Widgets usually enable the user to select a part of the webpage where he/she needs help, which are matched to a certain mechanism within the cloud. When a widget is triggered, the selected content of the webpage is wrapped into a request and then is forwarded to the extension, which sends it to the centralized cloud server. The cloud server handles the request, triggers the function or a

combination of functions in the cloud that simplifying the content according to the users preferences. After that, the request is sent back to the extension where it is rendered as a result within the webpage and thus presented to the user.

As seen in the workflow the system can be separated into two components. The first component deals with user interaction by offering an **adaptable personalized user interface** with widgets that trigger helper function. The second component deals with **adaptable personalization of content** by offering functions that convert content into a better to understand format and then present it in an individualized way to the user.

Adaptive/Adaptable personalized user interface component

The injected user interface consists of two parts:

- **User-Interface:** The user interface container as it is presented to the user
- **Widget:** Each user interface hosts widgets that trigger a function or a combination of functions of the framework

User Interface

To suit the requirements of different users, the framework will host many different kinds of user interfaces that can be configured for the individual user. User interfaces in the framework are developed in JavaScript and have to extend a base class. The inheritance makes user interfaces exchangeable and new user interfaces are developed with ease. When the webpage has fully loaded, it is injected into the webpage, instantiated, configured and finally presented to the user.

Currently two user interfaces were implemented:

Slide-in Interface

This interface is positioned absolute on a side position of the webpage and sliding in when the user would click on it (Figure 2). The advantages of this interface is that it does not consume too much room on the webpage itself, and is always visible. A big disadvantage, however, is that the interface might overlap with content of the original webpage.

The user interface is based on jQuery and the jQuery TabSlideOut plugin:
https://github.com/microneer/jquery.tabSlideOut.js/tree/master

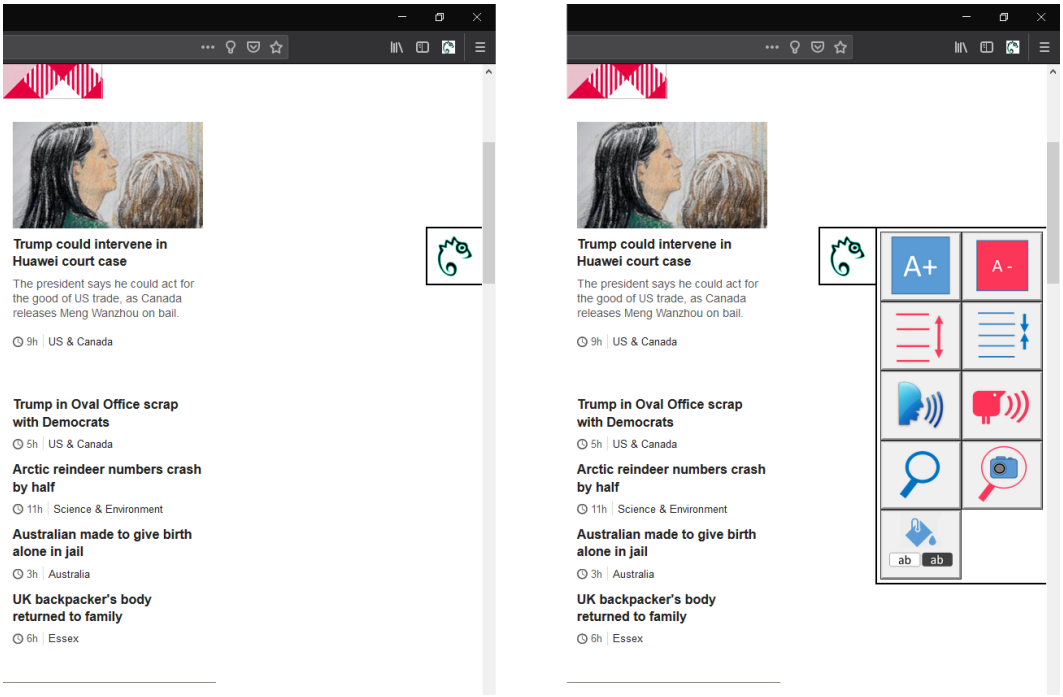

Figure 2: Slide-in user interface screenshot (left collapsed, right expanded)

Overlay Interface

This second interface was designed as an overlay (see figure 3) and was created with the jQueryUI library (https://jqueryui.com/).

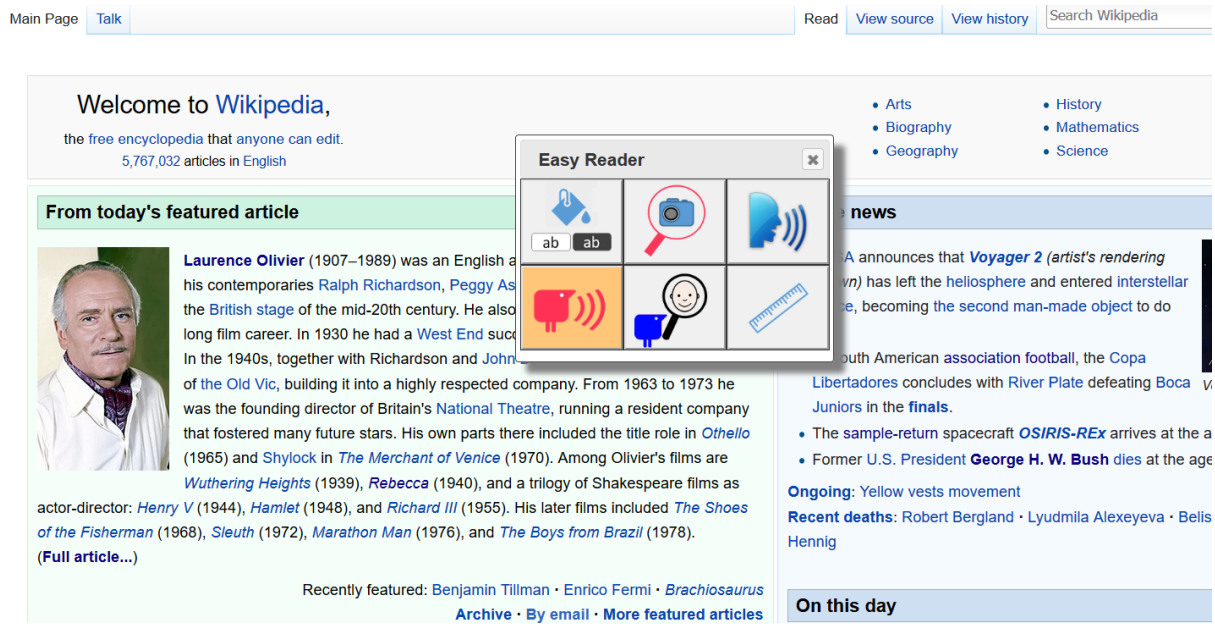

Figure 3: Overlay user interface screenshot

A disadvantage of overlay interface is that it might block the view on elements beneath it, but it can be moved through dragging. An advantage is that all functions are always visible with might be appropriate for some users.

Widgets

Each user interface can host widgets that allow users to trigger help provided by the framework. A widget in the framework can be developed with any valid HTML element, and is created within the initialization of the user interface. Like a user interface, widgets have to extend a base class, are developed in JavaScript that creates the desired HTML elements, and are styled with CSS. This inheritance makes widgets exchangeable for the individual user and new widgets can be developed and integrated into the framework with ease.

Widgets allow the user to select a certain part of the webpage where they need help an action.

Different widgets can utilize different HCI paradigms to select content. For example if a user wants to get help for a paragraph in a text that he does not understand, there could be multiple ways to select the paragraph:

Example 1

- Widget implemented as an On/Off button
- When the user selects the widget it changes appearance so that the user is notified that the widget is active
- Once the widget is active and the user clicks on a paragraph, the widget will get the content of the paragraph
- The paragraph is sent to the connected function in the cloud

Example 2:

- Widget implemented as a button
- Users have to mark the paragraph with their mouse
- When the user clicks the button the widget gets the marked paragraph on the webpage
- The paragraph is sent to the connected function in the cloud

Example 3:

- Widget implemented as an audio listener that listens for keywords spoken by the user
- When the user speaks the keyword the widget gets the paragraph under the current mouse position
- The paragraph is sent to the connected function in the cloud

These three examples should demonstrate that there are many different ways to achieve the selection of a paragraph within a webpage. First user tests have shown that there is a strong need from end users for the framework to support different HCI paradigms. In a test scenario, we asked users to test a Text-To-Speech function of the framework that could be triggered as described in Example 1 above. Users had to click on the widget in the user interface to activate it, and then would have to click on a paragraph in the content of a webpage to trigger the Text-To-Speech function that would read aloud the selected paragraph.

For some users this way of selecting a paragraph felt very natural and they had no problems performing the task. However, some users were struggling with this task, as they were used to a different HCI paradigm as described in Example 2. This is why the framework supports multiple

interaction paradigms and the architecture allows to integrate new ones with ease. The framework should adapt to the users' needs, and not the other way round.

Widgets always produce the input for the underlying functions or for a combination of functions that the framework offers. Currently the following input/output types are supported:

- Word: a single word within the content of a webpage
- Sentence: a single sentence within the content of a web-page
- Paragraph: a paragraph within the webpage
- Page: the content of an entire webpage
- Image: an image on a webpage
- Audio: either a link to an audio file/stream or binary audio data
- Video: either a link to an video file/stream or binary video data
- Void: as the name implies, this is an empty input type. This is used for functions that do not require any input, for example styling and layout functions that would not change the content. An example for that would be a function that would invert background colour with the font colour for better readability.

A single widget can support multiple input/output types and can be matched to any function that would accept this type.

Currently a widget gets information about the underlying function when it is initialized:

- Label: Label of the function
- Icon: A link to an icon for the function
- Description: A small description of the functionality of the function
- Long description: Detailed description of the functionality of the function
- Personal icon (Optional): A link to a user generated icon of

Widgets can use this information to describe the functionality that is triggered by the widget to the user either visually or in another way.

Adaptive/Adaptable content personalization

Adaptation and personalization of web-content within the EasyReading framework is performed on 3 levels:

- **Translation:** Understanding and simplification of the textual source into a different easier to understand format, like plain language, Easy-to-Read, or a reduction of language levels. Automated language understanding is a difficult technical area which requires fundamental knowledge in computational linguistics and AI. Currently first tools like the IBM Ability Lab Content Clarifier[1], which is designed to help simplify, summarize and enhance web content, emerge. Use case scenarios of such translations are for example word replacements, sentence reordering or simplification of content.
- **Annotation:** Enrichment of content through additional resources (symbols, pictures, videos, speech synthesis): Depending on the actual needs of the user the content of the original website must not only be translated into a language easier to understand but also needs to

[1] https://contentclarifier.mybluemix.net/

be enriched with additional information to ensure that the user can understand the content. An example would be the additional output of text by speech, where the currently spoken word or sentence would be highlighted in the web content.

- **Adaptation:** Structure and layout simplification of a webpage. Simple examples for a layout adaption would be the inversion of colour (e.g. white background and black font changes to black background and white font) for better readability. A more complex example for such an adaption would be the change of structure of a webpage so that only content is shown, and distracting elements like navigation, header, footer and sidebar are hidden. For this example to reliably work, the webpage has to specify the region of the content by either the proper HTML5 tag or the corresponding ARIA[2] attribute. The W3C personalisation taskforce[3] is currently working on new meta-tags and attributes that would allow content creators to identify common user interface elements, menus etc. within the HTML code. If webpages implement this meta-tags and attributes – the whole site can reliable be adjusted for the individual user's needs.

As descripted in the workflow the adaption of content within the EasyReading framework consists of two parts:

- **Functions:** Convert content into an easier to understand format, by applying one or a combination of the personalization mechanisms above.
- **Presentation:** Presentation of the result of a function to the end user.

Functions for content personalization

Functions within the EasyReading framework are implemented in JavaScript and are triggered by a widget. Like widgets, functions have to extend a base class, which makes it easier for other developers to extend the framework with new functions.

Functions accept one or more of the input/output types that a widget passes as a parameter when called. Functions use this input in combination with the user profile to personalize the content for the individual user. Currently the framework differentiates between two function types:

- **Remote Function:** a function that is performed in the cloud. This is usually for things that need computation power or are too complex to be calculated in the client. An example for that would be a Text-To-Speech function. I would take the text to be synthesized and then invoke the used Text-To-Speech endpoint to create the speech.
- **Local Function:** a function that is injected in the client. These are mostly layout functions etc. that do not require cloud computation or database connections etc.

Each function returns an input/output type, which can be used as input for another function or is used as input for the presentation of the result.

Presentation of results

After a framework function was triggered by an event, the result – if there is any – is rendered within the webpage by a presentation. Similar to a user interface, presentations have to extend a base class, are developed in JavaScript to create the needed HTML elements and are styled with CSS. This

[2] https://www.w3.org/TR/wai-aria/
[3] https://w3c.github.io/personalization-semantics/

inheritance makes it very easy for developers to contribute new presentations to the framework. Presentations of a framework result can be exchanged with other presentation as long as the new presentation supports the required input/output type of the matched function.

Here are some examples for presentations currently hosted by the framework:

Paragraph-Switcher

As the name implies, this presentation allows to render a paragraph as replacement for another paragraph. Figure 4 shows the original of a paragraph before the paragraph switcher rendered a result from a framework function. The framework function triggered another function from the IBM content clarifier that enriches paragraphs with AAC Symbols (Figure 5). By pressing the EasyReading logo at the bottom of the paragraph, users are able to switch back to the original and forth to the AAC-version of the paragraph (Figure 5 and Figure 6).

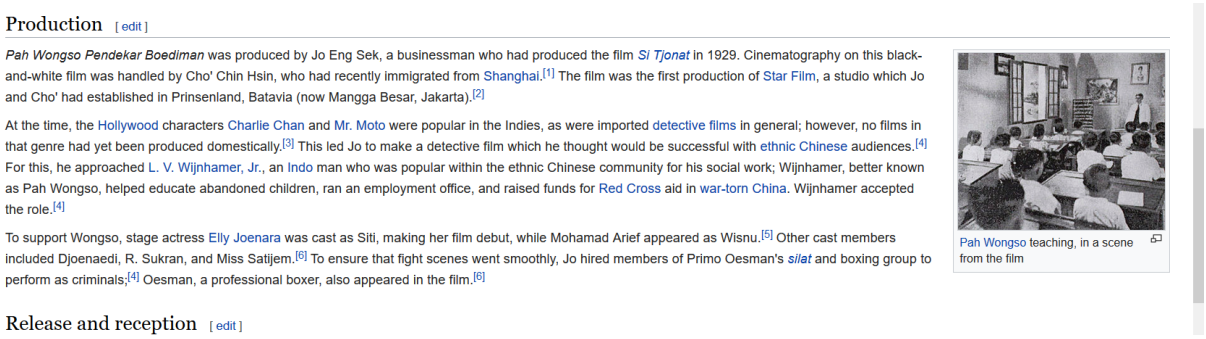

Figure 4: Paragraph before rendering of function result

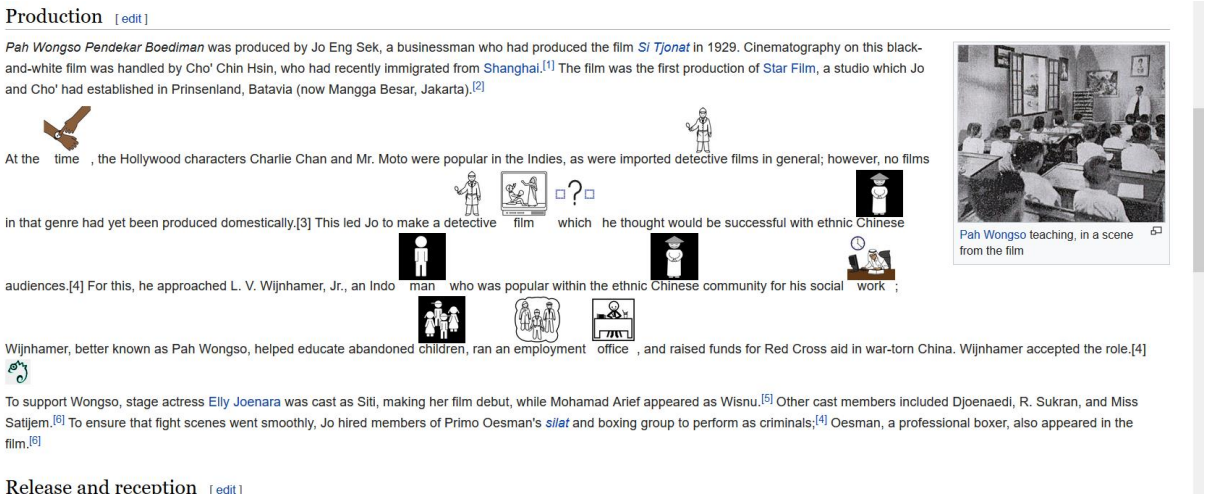

Figure 5: Paragraph after rendering of result

Figure 6: Paragraph switched back to original

Tooltip

A tooltip allows to render a sentence, a word, or an image. Once the tooltip has rendered the result within the webpage, it is marked as a dotted line. See word “worker” in Figure 7. Once the user hovers with the mouse over the dotted line, the tooltip is shown for the user (Figure 8).

Figure 7: Tooltip

Figure 8: Tooltip on mouse hover

Audio player with text highlighting

Figure 9 shows the audio player with text highlighting in action. It would mark the current spoken word and sentence so that it is easier for people with reading difficulties to see.

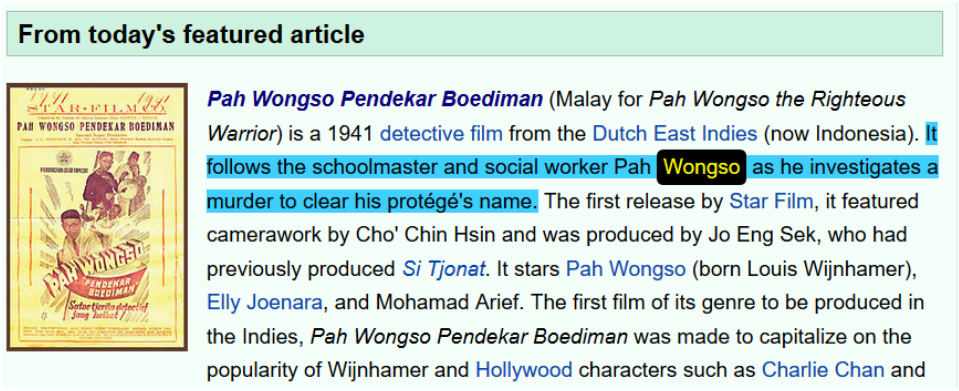

Figure 9: Audio player with text highlighting

Adaption Strategies used in the EasyReading Framework

In the introduction section, different types of adaption strategies were introduced. The EasyReading framework will support the following adaption strategies:

- **Adaptable with system support:** As carers should be able to adjust the user interfaces of their clients in the backend, the framework has to support this strategy. In this, carers can manipulate the user interface directly, and not only the user profile.
- **Adaptive with user control:** A fully adaptive user interface without user control might be confusing to many people. Without user control, the user interface and the way of content adaptation would change dynamically whenever the system detects changes in the profile of the user. This could lead to:
    - Dynamically added widgets and functions in the user interface
    - Dynamic removal of functions and widgets
    - Dynamic switch to another user interface
    - Dynamic change of presentation
    - …

  Consistency across webpages is very important for people with cognitive disabilities and also addressed in Guideline 2.1: Predictable of the W3C Web Content Accessibility Guidelines (WCAG) 2.1[4]. Therefore, a fully adaptive user interface without user control seems not appropriate for our target group at time of writing this report.

Adaptable with system support

As described in the previous chapters, the adaptations and personalization within the system can be done on user interfaces, widgets, functions, and the presentation of the result. To make adaptions persistent, they have to be stored in the database.

Currently all components within the framework (user interfaces, widgets, functions and presentations) specify data that they use for configuration and that is not part of the user profile with a JSON schema (https://json-schema.org/). With a JSON schema, the structure and format of JSON objects can be specified. The schema is used to automatically create the necessary table in the database when the system starts up. Besides the dynamically created tables, the database also hosts static tables that are used to link the dynamically created tables.

[4] https://www.w3.org/TR/WCAG21/#predictable

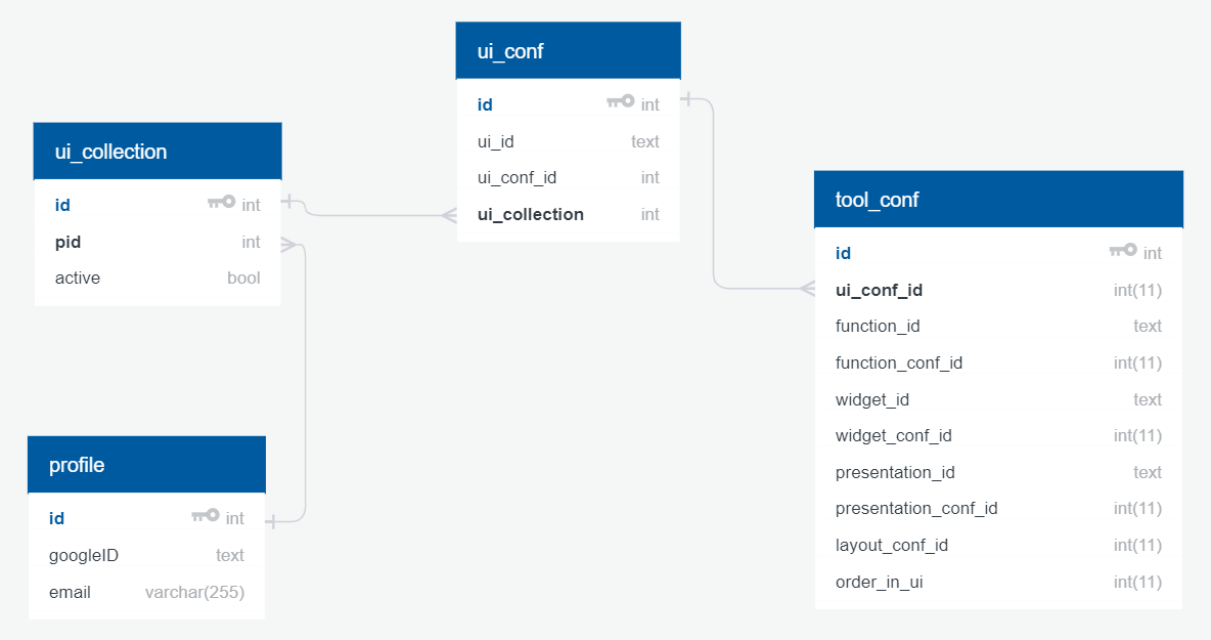

Figure 10: Simplified database schema of static tables

Figure 10 shows the database schema of the static tables, with a simplified user profile. Each user profile can have multiple UI-Collections (ui_collections) while only one can be active for a user at time. A UI-collection can be seen as a container for user interface configurations, allowing carers and users to try out different user interface configurations without losing the old configuration. Each UI-collection can hold many user interface configurations (ui_conf). With the field ui_conf.ui_id, the table holding the user interface configuration that was dynamically created by the JSON schema can be identified. With the field ui_conf.ui_conf_id, the row holding the configuration data within that table can be located. This is how the dynamic tables are matched with the static tables, and allow applications to fetch the configuration data for the individual user.

Each user interface configuration can have multiple tool-configurations (tool_conf). A tool is a combination of a widget, a function and a presentation.

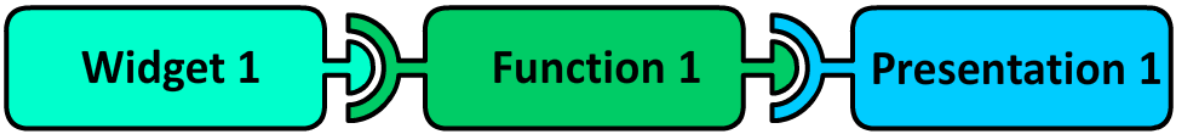

Figure 11: A tool within the EasyReading framework

Figure 11 visualizes the workflow of a tool. Widgets can be matched with a function that match their input/output type as described in earlier chapters. Functions take that input/output type and convert it to another or the same input/output type. After that, they pass the generated output to a presentation that accepts that input/output type.

Matching the static table for tool configuration (tool_conf) with the dynamically created tables of widgets, functions and presentation is similar to the user interface configuration. Links to the configuration tables for the widget, the function and the presentation are stored in the tool_conf table, as well as the id of the according row in that configuration table. Currently a tool can only hold

one function. However as described above, it is planned to also allow a combination of functions (see figure 12).

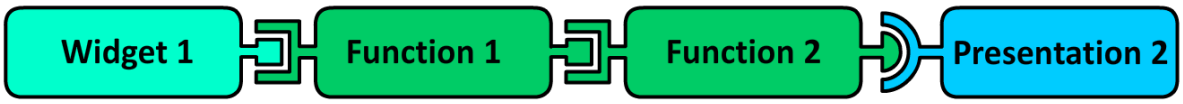

Figure 12: Combination of functions within a tool

This enables the framework to build new customized tools for the individual user without any programming effort. For example, widget 1 could be a widget that allows to select a single word within the webpage. Function 1 could be a dictionary function that accepts a word, and translates it to another language. Function 2 could be a text to speech function that accepts the translated word as input and generates audio as output. Presentation 2 could be an audio player that accepts audio as input. The resulting tool would be a dictionary for single words that reads the translated word aloud.

Forms for adaption

To configure a user interface collection for a user, forms in the backend and in the clients are needed. These store the configuration data in the according data structure. Creating the forms by hand is tedious. Therefore, the JSON schema is utilized again to automatically create those forms:

- JSON Schema to HTML form generator: http://brutusin.org/json-forms/
- React JSON Schema Forms: https://mozilla-services.github.io/react-jsonschema-form/

This should also attract external developers, as they are able to concentrate on the functionality of their components to add, and not on the configuration. Besides that, all the forms for configuring components have the same look and feel. If external developers really need to implement their own form, they can override this automatic mechanism.

By utilizing the JSON schema, only forms for the static tables like the UI-Collection configuration are needed. Currently, at time of writing this deliverable, these forms are in development.

If time allows it, a configuration wizard is planned, to help new users to create an optimized user interface and optimized adaption of content.

Adaptive with user support

The first step after making the user interface and the content personalization adaptable is to make it adaptive. The primary goal is to use or adapt an existing system for adaptive user interfaces and integrate it with the EasyReading framework. Peissner[8] gave a good overview on related research projects that developed adaptive systems. In this paper, Peissner also identified three key requirements for the market acceptance of adaptive user interfaces system [8]:

- Efficiency in the development process of adaptive user interfaces
- Short learning curve for developers
- Comprehensible and controllable development processes for generating resulting user interfaces

Based on this key requirement, the following requirements for selecting an existing adaptive system to be integrated in the EasyReading framework have been defined:

- The system must be node.js compatible: Currently the whole architecture is based on a single node.js application, making it extremely easy for external developers to run and test the system locally. Selecting an existing solution that would require additional runtimes and software to install, would increase the complexity of the whole system.
- The system must be mature enough and have a stable API
- The system should provide public available adaption mechanisms and should be open to external contributions

Currently existing projects like MyUI[5], GPII[6] or the OASIS[7] have been briefly evaluated in respect to the requirements mentioned above.

At time of writing this deliverable, using the GPII or parts of GPII for seems the most promising approach for the EasyReading project. The architecture of the GPII as developed by the Cloud4all project uses an ontology of user needs and preferences that are directly linked to user interface and interaction aspects [9]. The linking is done with a rule based matchmaker (RBMM) that matches user preferences and needs, with solutions available to the system and settings supported by the solution. The matchmaker results therefore in a fully configured solution on the specific system based on the individual preferences and needs of a user.

To make this work within the EasyReading framework, the profile has to be conform to the user needs and preferences defined by GPII. After that, the components of the easy reading framework have to be matched with solutions offered by GPII and the settings of the components have to be matched with the settings supported by GPII solutions. This should enable the RBMM to create a fully configured and optimized user interface based on the profile at runtime.

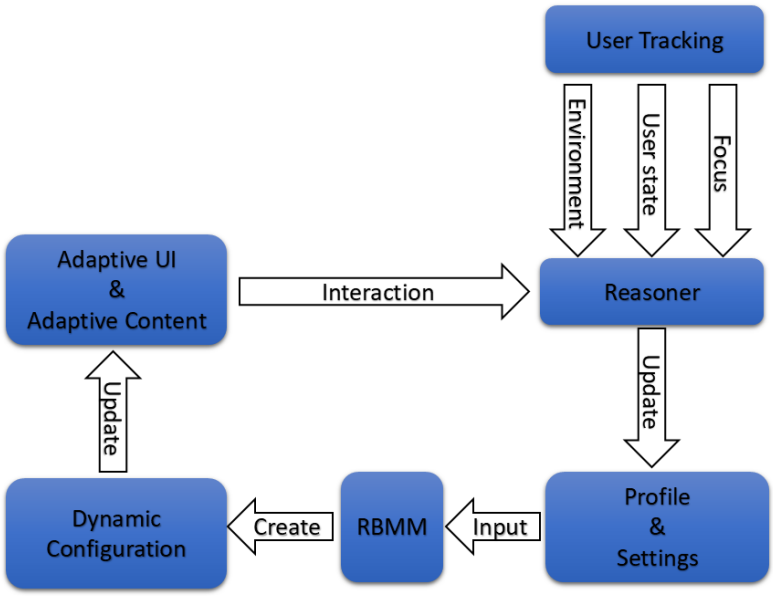

Figure 13: RBMM of GPII integrated in the EasyReading framework

[5] http://www.myui.eu/
[6] https://gpii.net/
[7] http://www.oasis-project.eu/

Figure 13 shows a possible integration of GPII within the EasyReading framework and the resulting workflow. Assuming a valid profile and settings are present in the system, once the user logs in, the RBMM will create a dynamic configuration that is used to initialize the adaptive personalized user interface component and the adaptive personalized content component. While the user is interacting with the client, the user tracking component is used to generate data about the current environment of the user, the element on the webpage currently focused by the user, and the current cognitive state of the user. Based on the user interaction and user tracking, a reasoner updates the user profile and preferences in real time. Whenever an update in the profile and preferences is detected, the RBMM will create a dynamic configuration. If it differs from the old configuration, the user will be informed about the change. If the user agrees, the change will be performed.

Another advantage of using GPII for the EasyReading framework is that users could import their preferences from GPII and use it within the EasyReading framework.

Conclusions

The consortium is currently working to make the system fully adaptive, while still evaluating and validating which existing systems that realize adaptive user interfaces should be used for the EasyReading framework. As mentioned above, GPII is at the moment favored, as it is one of the most recent and popular projects and because it could be seamlessly integrated within the node.js environment.

References

[1] Benyon, D. I. (1987). System Adaptivity and the Modelling of Stereotypes. National Physical Laboratory, Division of Information Technology and Computing

[2] Hook, K., 1998. Evaluating the utility and usability of an adaptive hypermedia system. Knowledge Based Systems 10 (5), 311–319.

[3] Brusilovsky, P., Maybury, M.T.: From Adaptive Hypermedia To The Adaptive Web. Communications of the ACM 45(5), 31–33 (2002)

[4] Talia Laive (2010). Benefits and costs of adaptive user interfaces. Int.J. Human Computer Studies 68

[5] Gullà, Francesca & Ceccacci, Silvia & Germani, Michele & Cavalieri, Lorenzo. (2014). Design Adaptable and Adaptive User Interfaces: A Method to Manage the Information. Biosystems and Biorobotics. 11. 10.1007/978-3-319-18374-9_5.

[6] Peissner, Matthias & Schuller, Andreas & Spath, Dieter. (2011). A Design Patterns Approach to Adaptive User Interfaces for Users with Special Needs. 6761. 268-277. 10.1007/978-3-642-21602-2_30.

[7] Petz, Andrea. (2013). CAPKOM: A Wizard to Facilitate the Web Experience of Users with Cognitive Disabilities. 10.3233/978-1-61499-304-9-976.

[8] M. Peissner, A. Schuller, D. Ziegler, C. Knecht, and G. Zimmermann, “Requirements for the Successful Market Adoption of Adaptive user interfaces for Accessibility,” in Universal Access in HumanComputer Interaction. Design for All and Accessibility Practice, ser. Lecture Notes in Computer

Science, C. Stephanidis and M. Antona, Eds. Springer International Publishing, 2014, no. 8516, pp. 431–442

[9] Madrid, J., Peinado, I., Koutkias, V.: Cloud4all Priority applications and User Profile Ontology (D101.1). Public Deliverable of the Cloud4all Project (2012), http://cloud4all.info/render/binarios.aspx?id=90