



## SEVENTH FRAMEWORK PROGRAMME INFORMATION AND COMMUNICATION TECHNOLOGIES

### Project

Accessibility Assessment Simulation Environment for New  
Applications Design and Development  
(ACCESSIBLE, Grant Agreement No. 224145)



### Deliverable

D 4.2 – A software package containing a set of modelling tools, rules inference engine, and the rules graphical editor

<b>Lead beneficiary</b>	FFCUL
<b>WP. no, title and activity type</b>	WP4 – Services and Content Knowledge Infrastructure
<b>Contributing Task (s)</b>	T4.2 – Accessibility-centred Rules Inference Engine T4.3 – Rules User Interface
<b>Dissemination level</b>	PU - Public
<b>Delivery date</b>	April 2010
<b>Status</b>	Update
<b>File name and size</b>	"D4.2_updated_version_ACCESSIBLE_FFCUL_WP4-FD04_2011_v01.doc.doc", 1.6 MB



*(This page has been left empty in purpose.  
The same stands after each main section.)*

## Authors List

Leading Author ( <i>Editor</i> )				
	<i>Surname</i>	<i>Initials</i>	<i>Beneficiary Name (Short Name)</i>	<i>Contact email</i>
	Lopes	R.	FFCUL	rlopes@di.fc.ul.pt
Co-authors ( <i>In alphabetic order</i> )				
#	<i>Surname</i>	<i>Initials</i>	<i>Beneficiary Name (Short Name)</i>	<i>Contact email</i>
1	Lösch	E.	USTUTT	Eva.Loesch@iao.fraunhofer.de

## Peer Reviewers List

#	<i>Surname</i>	<i>Initials</i>	<i>Contact email</i>
1	Van Isacker	KVI	projects@marie-curie-bg.org
2	Garrè M.	SOFTECO	marco.garre@softeco.it

## History Table

Date	Comments
April 2010	Final version delivered to EC
February 2011	Updated version
April 2011	Finalise the update version



## Executive Summary

This is the updated version of the Deliverable 4.2 that was issued in April 2010 and incorporates the comments that were received by EC reviewers during the 2<sup>nd</sup> review meeting.

This deliverable presents the core software components of ACCESSIBLE's Services and Content Knowledge Infrastructure: the Rules Inference Engine, and the Rules User Interface.

The goal of these components lay at providing a way to infer accessibility knowledge (e.g., which accessibility tests are appropriate for a given disability), which can be used by the accessibility assessment tools (such as those being developed in WP5), in order to deliver a semantic accessibility assessment mechanism and providing to the tools the relevant inferred knowledge from the ACCESSIBLE ontologies.





## Table of contents

Authors List .....	4
Peer Reviewers List .....	4
Executive Summary .....	7
Table of contents .....	9
List of figures .....	11
List of definitions & abbreviations .....	13
1 Introduction .....	14
2 Rules Inference Engine .....	15
2.1 Introduction and Context .....	15
2.2 Requirements .....	16
2.3 Usage Scenarios .....	18
2.4 The Inference Engine .....	19
2.5 Implementation .....	21
2.6 Application Programming Interface (API) .....	24
3 Rules User Interface .....	27
3.1 Introduction .....	27
3.2 Requirements .....	28
3.3 The Rules User Interface .....	30
3.3.1 The Graphical User Interface .....	31
3.3.2 The Ontology Management Module .....	34
3.3.3 The Search Module .....	34
3.4 Implementation .....	35
4 Conclusions .....	37
References .....	38



## List of figures

Figure 1 Architecture of the ACCESSIBLE Rules Inference Engine .....	19
Figure 2 UML Sequence Diagram for the Rules Inference Engine.....	20
Figure 3 Rules Inference Engine execution algorithm .....	23
Figure 4 WSDL service for Public API of the Rules Inference Engine .....	24
Figure 5 Bindings for Namespaces and End Points.....	25
Figure 6 Integrating a set of bindings in a software component.....	26
Figure 7 Use Case diagram of the ACCESSIBLE Rules User Interface.....	29
Figure 8 Architecture of the ACCESSIBLE Rules User Interface .....	30
Figure 9 Start screen of the Rules User Interface .....	31
Figure 10 Screen for creating a new accessibility standard.....	32
Figure 11 Screen for creating a new accessibility guideline.....	32
Figure 12 Screen for specifying or editing an own accessibility assessment rule .....	33
Figure 13 Example accessibility assessment rule in SWRL format .....	36



## List of definitions & abbreviations

*(in alphabetic order)*

API	Application Programming Interface
JVM	Java Virtual Machine
OWL	Web Ontology Language
RDF	Resource Description Framework
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Query Language for RDF
SWRL	Semantic Web Rules Language
URI	Uniform Resource Identifier
WCAG2.0	Web Content Accessibility Guidelines 2.0
WSDL	Web Services Description Languages

# 1 Introduction

This report presents the details of the core development tasks of Work Package 4, *Services and Content Knowledge Infrastructure*. These tasks are T4.2, *Accessibility-centred Rules Inference Engine*, and T4.3, *Rules User Interface*. The main goal of these tasks is the creation of a set of components that help accessibility assessment tools on managing and interacting with the ACCESSIBLE knowledge base (as detailed in D4.1), in order to allow for tailored accessibility assessments to be performed efficiently.

The next Sections of this deliverable will detail on the particular aspects each one of the two components of the ACCESSIBLE architecture developed in this Work Package.

## 2 Rules Inference Engine

This Section presents the details of the ACCESSIBLE Rules Inference Engine component, a core component of the ACCESSIBLE system architecture. Its goal lays on providing reasoning facilities to the different accessibility evaluators, as well as to the ACCESSIBLE portal. Next, we begin introducing the particular aspects of the Rules Inference Engine, and the context upon which it is inserted.

### 2.1 Introduction and Context

Accessibility means different things to different people. For some accessibility experts it represents the *ability to access* a resource, a piece of information, or to perform an interaction in order to fulfil a given goal and/or task. Others see it at a more overarching level, where *accessibility* strives for the study of the *user experience* conveyed to users with disabilities, au pair with what is experienced by non-disabled users. This later definition goes hand in hand with the four principles of WCAG 2.0: **Perceivable**, **Operable**, **Understandable**, and **Robust**. WCAG 2.0 centres on these principles as the core user experience factors that allow for a Web site to be accessible for users with disabilities, in the following way:

- *Perceivable*: This means that users must be able to perceive the information being presented (it can't be invisible to all of their senses);
- *Operable*: This means that users must be able to operate the interface (the interface cannot require interaction that a user cannot perform);
- *Understandable*: This means that users must be able to understand the information as well as the operation of the user interface (the content or operation cannot be beyond their understanding);
- *Robust*: This means that users must be able to access the content as technologies advance (as technologies and user agents evolve, the content should remain accessible).

However, from the perspective of each user with some sort of disability, their personal limitations and, consequently, their *inability to access* are paramount. On the other hand, companies willing to deliver products and services that are accessible might not be capable of coping with the rich diversity and profound complexity of accessibility compliance, thus requiring a compromise on partial compliance.

For all of these reasons, *accessibility* is a dynamic concept that shifts from person to person, from company to company. One of the goals of the ACCESSIBLE project lies on supporting accessibility for all of these actors. The first step to achieve this goal is accomplished by the ACCESSIBLE knowledge infrastructure, by the means of a set of ontological concepts and rules (specified in the OWL and SWRL languages) that cover a big part of the spectrum of accessibility in different technological domains. However, to cope with the different requirements of accessibility-related scenarios, the correct slicing of this knowledge is important for their success.

The goal of the Rules Inference Engine is, in the light of this context, to provide support to tailored accessibility knowledge, tailored accessibility evaluation, and ultimately to leverage the momentum for the creation of accessible software products and services.

Based on the presented assertions, the next Section details the requirements that must be taken into account for the design and implementation of the Rules Inference Engine.

## 2.2 Requirements

A correct and useful Rules Inference Engine must cope, right from the start, with several functional and non-functional requirements. In deliverable D2.2b we have detailed the requirements that must be taken into account to devise the architecture, design, and implementation of the Rules Inference Engine.

In this Section we re-iterate some of the requirements for the ACCESSIBLE Ontological Knowledge Resource, in the light of the ones that are greatly important for the Rules Inference Engine. For each one of the requirements, a brief discussion will provide clarifications for the decisions made for the Rules Inference Engine.

**G-REQ3-1:** The ontology will support the OWL Web Ontology Language (OWL DL and OWL Lite).

**G-REQ3-2:** The ontology shall support the SWRL (a Semantic Web Rule Language) rules, which form an implication between an antecedent (body) and consequent (head).

**G-REQ3-3:** The ontology shall support SPARQL queries that consist of triplets, in order to narrow the information space of accessibility assessment according to specified semantics of usage scenarios.

These first requirements provide the technological frame upon which the Rules Inference Engine lies. By applying OWL, SWRL, and SPARQL, the entire spectrum of knowledge access, manipulation and integration will be centred on the Semantic Web technological domain.

**G-REQ3-14:** The execution of queries and rules shall be supported through the integration of an open source inference engine that shall support the usage of SPARQL as well as SWRL.

This requirement provides an additional view on how the Rules Inference Engine must be architected and implemented. While the Semantic Web aspect is enforced once again (like in the previous set of requirements), there is an extra requirement for the licensing model of core components. The Rules Inference Engine must be based on existing open source components, in order to be possible to freely integrate it into ACCESSIBLE's evaluator components.



**P-REQ3-1:** Response times in communication of the ACCESSIBLE knowledge resource with the assessment system shall be in line with those specified in the communication protocol selected.

**P-REQ3-2:** In the same way, the inference engine in the complex rules and queries execution shall also take into account the performance.

Regarding performance, the Rules Inference Engine must use state-of-the-art communication features with the ACCESSIBLE knowledge resource and, ultimately, not hinder the performance of the different accessibility evaluators.

**R-REQ3-1:** The ontology-based resource shall provide answers just for the knowledge embedded in ontologies (both general and domain-specific). All knowledge not represented will be regarded as unknown, not as false (i.e. open-world assumption).

This requirement enforces the Rules Inference Engine to be reliable in respect to the limits of the knowledge base. Therefore, the selected technologies must be capable of being configured to work on an *open-world assumption*.

**M-REQ3-3:** The knowledge inference engine must be interoperable and cope with open technologies. For this, it will be developed using open source knowledge software components developed in portable languages (e.g. Java).

This requirement is of significant importance in the definition of both the architecture and implementation aspects of the Rules Inference Engine. The selected technologies must be open, work on different Operating Systems, and easily articulate with other technologies. Consequently, the Rules Inference Engine must provide architectural solutions for its integration and usage in different usage scenarios.

**M-REQ3-4:** In this same way, the ACCESSIBLE knowledge resource and the developed inference engine as well as the rules editor shall be enough documented in order to be understandable for users external to the development.

Since the Rules Inference Engine by itself has little to no use, its integration in other technologies, products, and services is crucial. For this to successfully happen, the previous requirement must be accomplished. However, it is also important to provide proper documentation (technical, tutorials, guides, etc.) on how to integrate the Rules Inference Engine outside the scope of the ACCESSIBLE project. The integration of this component in other ACCESSIBLE components (such as the accessibility evaluation tools) provides a first solid ground on this issue. While this deliverable (D4.2) details the implementation aspects of the Rules Inference Engine, these aspects will be further detailed in the deliverable D4.3 [1].

**G-REQ1-9:** The system shall support any different potential users (developer, designer, IT manager, etc.) to test its developments against any set of selected accessibility criteria and/or rules specified by users.

**G-REQ1-10:** The ACCESSIBLE system shall support the implementation of appropriate rules based on the ACCESSIBLE knowledge resource and on the user's preferences.

**G-REQ1-11:** The ACCESSIBLE system shall support personalised accessibility assessment functionalities that must cope with the user's disabilities and individual's preferences.

These three requirements are centred on the personalisation aspect of accessibility assessment, which was also discussed previously in the Introduction Section. The Rules Inference Engine must take into account that personalisation (and other knowledge slicing tasks) will result in different evaluation procedures and, ultimately, in different accessibility evaluation outcomes. Thus, there must be a set of features both at the architectural and implementation levels that provide these affordances.

These requirements have a deep impact on the way the Rules Inference Engine is architected and implemented. Since these factors influence the way it is going to be used across the ACCESSIBLE architectural components, the Rules Inference Engine must be analysed from the perspective of its usage. The next Section presents a set of usage scenarios that will dictate the architecture and design of the Rules Inference Engine.

## 2.3 Usage Scenarios

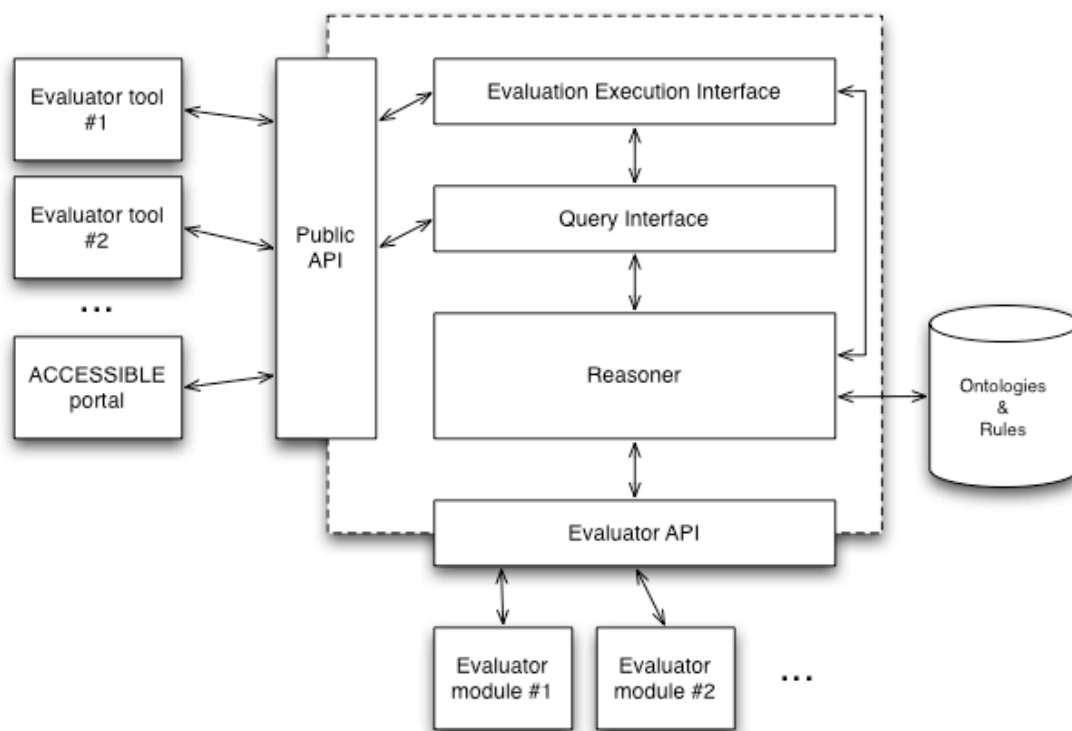
The previous Section presented the major requirements that must be taken into account when defining the Rules Inference Engine's architecture and corresponding implementation. From these requirements, we have extracted the following set of usage scenarios that are envisioned to be supported by the Rules Inference Engine:

- 1) *Browsing and querying the ACCESSIBLE knowledge interface.* Users must be capable of interacting with the knowledge resource in different ways, through different user interfaces, different tools, etc.;
- 2) *Selecting accessibility evaluation tests.* Users must be capable of selecting different accessibility evaluation tests through browsing and querying activities;
- 3) *Triggering accessibility evaluations.* Lastly, the Rules User Interface must not hinder the execution of tailored accessibility evaluations. Therefore, it must provide features for its integration into evaluation procedures;
- 4) *Extending and integrating capabilities.* Developers must be allowed to extend and use the Rules Inference Engine in miscellaneous situations. This will open the way to an increased awareness and success of the ACCESSIBLE project after its conclusion.

With the usage scenarios defined in this Section, and based on the requirements presented previously, the next Section details the Rules Inference Engine itself, particularly the architectural and design decisions that have been made.

## 2.4 The Inference Engine

The Rules Inference Engine's architecture was devised according to the constraints and requirements presented in the previous Sections. This architecture is presented next, on Figure 1:



**Figure 1 Architecture of the ACCESSIBLE Rules Inference Engine**

The main goal of the Rules Inference Engine is to provide a set of reasoning features to the different components of the ACCESSIBLE architecture. To accomplish this, the architecture of the Rules Inference Engine comprises the following internal components:

- 1) *Reasoner*: This component provides the core feature of the engine, which is the interpretation of ontologies and rules according to different criteria (e.g., a query), as well as to afford triggering tailored evaluation procedures;
- 2) *Query Interface*: This component implements querying features (e.g., through SPARQL queries) that are executed by the *Reasoner*. These queries allow the correct dicing of the ACCESSIBLE knowledge infrastructure (i.e., the ACCESSIBLE ontologies), which can be used afterwards for different tasks;

- 3) *Evaluation Execution Interface*: This component leverages the capabilities of the *Query Interface* component, in order to trigger tailored evaluation procedures on the *Reasoner* component.

These three core components of the Rules Inference Engine are responsible for the execution of the features that must be provided to external tools. Therefore, the way they interact is of the utmost importance. Based on the usage scenarios presented on the previous Section, we have defined two different execution modes for the Rules Inference Engine, as detailed in the following UML sequence diagram:

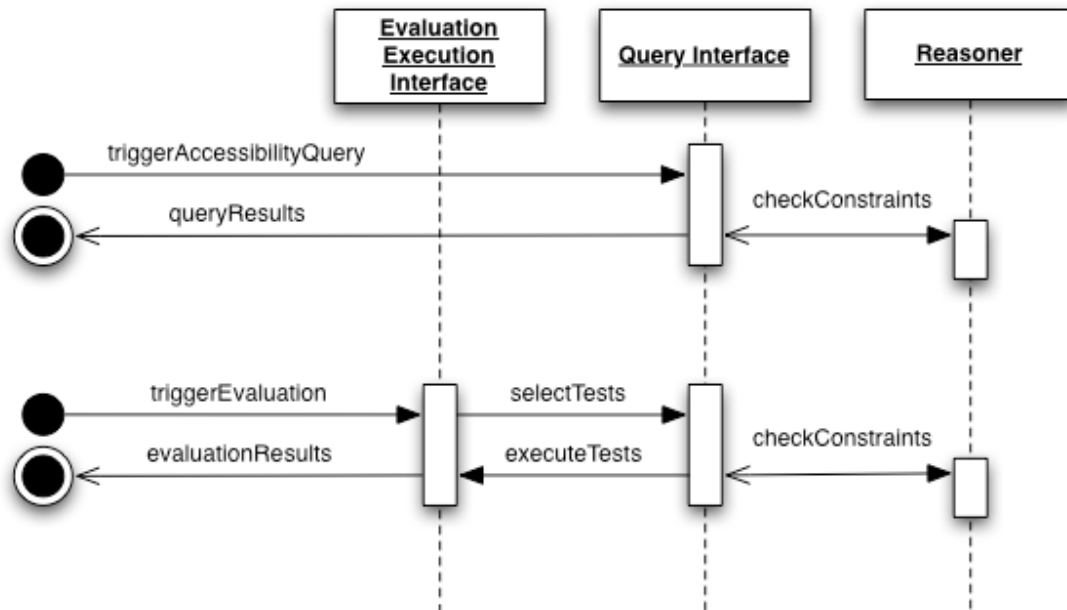


Figure 2 UML Sequence Diagram for the Rules Inference Engine

The first execution mode concerns the querying for accessibility knowledge. Here, the usage scenarios *Browsing and querying the ACCESSIBLE knowledge interface* and *Selecting accessibility evaluation tests* are covered by the first operation sequence depicted above.

The second execution mode provides the necessary functionalities to cover the *Triggering accessibility evaluations* usage scenario. In conjunction with implemented evaluation modules, this mode takes care of all interactions between evaluators and the *Reasoner*.

Consequently, in order to properly shield the features of the Rules Inference Engine from overly complicated programmatic interfaces, this component makes two APIs (Application Programming Interfaces) available. The goal of these APIs lays at the integration of the Rules Inference Engine in the different ACCESSIBLE architectural components. The APIs made available by the architecture are:

- 1) *Public API*: This API provides facilities for any component wishing to interact with the *Query Interface* (e.g., to extract some specific knowledge from the ontologies), or to trigger evaluations performed by a given evaluator component that has been connected to the Rules Inference Engine, through interfacing with the *Evaluation Execution Interface*;

- 2) *Evaluator API*: At a lower lever, the Rules Inference Engine can also be used as a central hub for evaluation components. In order to accomplish this task, the engine provides this API to existing (or future) accessibility evaluators.

Finally, the Rules Inference Engine is of little to no value if it does not interact with some external components. Therefore, its architecture reflects the integration of two types of external components:

- 1) *Ontologies & Rules*: Ontologies and rules form the core accessibility knowledge that is interpreted by the *Reasoner* component are imported directly from the ACCESSIBLE knowledge infrastructure (as detailed in D4.1 and accompanying software). Consequently, all interactions with the Rules Inference Engine triggered by external components are indirectly dependent on this knowledge base;
- 2) *External modules and tools*: Finally, the actual evaluation procedures are paramount to a successful use of the Rules Inference Engine. Therefore, modules and tools are expected to interact with it in different ways. For instance, an evaluation tool might require extracting some specific knowledge about accessibility evaluation in a particular technological domain (i.e., through the *Public API* component). Other modules can be directly plugged into the Rules Inference Engine through the *Evaluator API*, thus allowing their immediate usage by other tools.

Based on the architecture and design decisions presented in this Section, we detail their reflection on the implementation of the Rules Inference Engine.

## 2.5 Implementation

The Rules Inference Engine has been implemented on top of already existing software components, particularly by leveraging state-of-the-art open-source work on ontology and rules management and execution. The following six main components and concepts were used as the base for the Rules Inference Engine:

- 1) *JVM (Java Virtual Machine) technologies and libraries*: the foundation of the implementation of the Rules Inference Engine is the JVM and related libraries. We opted for this technology due to its pervasiveness (available in all relevant execution platforms) and openness (licensed as open-source). Furthermore, the software libraries that are made available right from the start by the JVM provide a lower entry barrier in the development of software components;
- 2) *JVM meta-programming*: since one of the goals of the Rules Inference Engine is to support an easy integration of external evaluation modules (i.e., the *Evaluator API*), meta-programming techniques such as run-time class inspection provide the constructs for an easy extensibility API of the engine;

- 3) *Groovy programming language and libraries*: Groovy<sup>1</sup> is an open-source, agile programming language (akin to Ruby, Python, etc.) that sits on top of the JVM, providing a set of high abstractions and concepts – both at the language and libraries levels – to facilitate readability and flexibility of implementing complex algorithms and software components. Consequently, we opted for the usage of this language to implement the Rules Inference Engine;
- 4) *Jena Semantic Web Framework*: Jena<sup>2</sup> is an open-source, state-of-the-art Java library for building Semantic-Web applications. It provides support for several ontology formats, such as RDF, RDF-S, and OWL, as well as a set of querying interfaces based on SPARQL. This library fits perfectly on the goals of the Rules Inference Engine, which has been used in the core implementation of *Reasoner* and *Query Interface* components;
- 5) *Pellet*: Pellet<sup>3</sup> is an open-source, state-of-the-art OWL 2 reasoner, made available as a Java library. It provides a set of constructs to parse, create, and trigger rules specified in different languages, such as SWRL. Pellet's APIs allow for its integration into different Semantic Web frameworks, such as Jena. Consequently, this library is used in the Rules Inference Engine as a way to integrate SWRL-based reasoning facilities, particularly on the *Reasoner* component of the engine;
- 6) *GroovyWS*: Since it is paramount to provide external access and integration facilities of the Rules Inference Engine, we have implemented a Web Services based API on the engine. For this task, we have leveraged GroovyWS<sup>4</sup>, an external library of the Groovy ecosystem, which provides a simple way to publish Web Services based on the specification of public classes. This library already includes bootstrap facilities (e.g., embedded Web server, WSDL generator, etc.), thus lowering the burden of externalising the Rules Inference Engine's feature set.

One novel contribution of the Rules Inference Engine concerns the interaction between the *JVM meta-programming* concepts and the *Jena* and *Pellet* libraries. The binding between them allows for a new way to specify accessibility assessment evaluations in such a way that is both generic and easily extensible. Consequently, we have implemented an execution algorithm in the core of the Rules Inference Engine that provides this functionality out-of-the-box as presented next on Figure 3:

---

<sup>1</sup> <http://groovy.codehaus.org/>

<sup>2</sup> <http://jena.sourceforge.net/>

<sup>3</sup> <http://clarkparsia.com/pellet/>

<sup>4</sup> <http://groovy.codehaus.org/GroovyWS>

```

Require:  $w \leftarrow$  Web page
Require:  $g \leftarrow$  Guideline specification in OWL
Require:  $c_g \leftarrow$  Class implementing guideline
Require:  $q \leftarrow$  Semantic query in SPARQL
Ensure:  $map_R \leftarrow$  Tests results
  {Initialisation of variables}
   $o \leftarrow$  generic ontology and mapping rules
   $o \leftarrow o \cup g$ 
   $map \leftarrow \emptyset$ 
   $i \leftarrow \text{instance}(c_g)$ 
   $i_{ns} \leftarrow \text{annotation}(i, \text{BindNamespace}, \text{namespace})$ 
  {Determine mapping between URIs and methods}
  for all  $m \in \text{methods}(i) \wedge \exists \text{annotation}(m, \text{BindTest},$ 
     $\text{endpoint})$  do
     $m_{ep} \leftarrow \text{annotation}(m, \text{BindTest}, \text{endpoint})$ 
     $m_{ns} \leftarrow \text{annotation}(m, \text{BindTest}, \text{namespace}) \vee i_{ns}$ 
     $uri \leftarrow m_{ns} + m_{ep}$ 
     $map[uri] \leftarrow m$ 
  end for
  {Get test set from SPARQL query and execute it}
   $\text{testset} \leftarrow \text{sparql}(o, q)$ 
  for all  $t \in \text{testset} \wedge \exists map[t]$  do
     $map_R[t] \leftarrow \text{invoke}(map[t], i, w)$ 
  end for
return  $map_R$ 

```

**Figure 3 Rules Inference Engine execution algorithm**

This algorithm takes as its input a tuple  $\{w, g, c_g, q\}$  and returns the map  $map_R$ , comprising tests names and their respective execution outcome. The execution plan is sub-divided into three stages:

- 1) *Initialisation.* It begins by bridging the ACCESSIBLE ontology represented by  $g$ , and initialising a map for method/URI bindings. Afterwards, an instance of  $c_g$  is created,  $i$ , with the JVM reflection method *instance*, and the default namespace for the guideline with the JVM reflection method *annotation* is extracted from this instance;
- 2) *Mapping.* An iteration is performed on all methods of  $i$  with the JVM reflection method *methods*, but only on those methods properly annotated with *BindTest* and its corresponding *endpoint* property. For each of those methods in the iteration, we extract both *endpoint* and *namespace* properties, defaulting to the instance namespace when the latter is not present. Afterwards, we create the appropriate method URI by concatenating the namespace with the end point and register a binding between this URI and the method;
- 3) *Testing.* The final stage of the algorithm performs the actual execution of the semantic accessibility assessment. This is done by extracting an appropriate set of tests from the ACCESSIBLE ontology corresponding to the semantics specified in the query  $q$ , ensuring that all of these tests are present in the method/URI bindings. Afterwards, each corresponding method is executed against  $w$  (the resource being evaluated, e.g., a Web page, a WSDL file, etc.), with the reflection method *invoke*. The execution outcome is added to the set of test results  $map_R$  for later processing.

The way each test must be specified in order to work with this algorithm is further detailed in the next Section. The implementation of the Rules Inference Engine by

itself has little to no use if it's not integrated into other ACCESSIBLE architecture components (e.g., standalone tools, portal, etc.) The next Section presents a first draft version of the public APIs defined by the Rules Inference Engine.

## 2.6 Application Programming Interface (API)

As presented in Section 2.4, the Rules Inference Engine provides two APIs to developers. The first API, the *Public API*, defines a set of constructs for a simple integration of the engine in external applications.

We have implemented a Groovy class that hides all internal interactions between the different components of the Rules Inference Engine. The methods defined in this class define the *Public API* for the engine, which must be easily integrated into ACCESSIBLE tools, portal, etc. Consequently, since each tool might be implemented in different platforms/frameworks/programming languages/environments, we opted to expose it as a WebService. For this task, we have used the *GroovyWS* component (presented in the previous Section), which helps transforming a Groovy class into a WSDL-based WebService. This WebService can be interacted through the SOAP protocol, which is the standard way of integrating this type of services. All modern programming languages provide library support to interact with SOAP-based Web services. Next, on Figure 4, we present the resulting WSDL specification:

```

83 </wsdl:output>
84 </wsdl:operation>
85 </wsdl:portType>
86 <wsdl:binding xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name="WebServiceSoapBinding" type="tns:WebServicePortType">
87   <soap:binding xmlns:soap="http://schemas.xmlsoap.org/soap/" style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
88   <wsdl:operation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name="create">
89     <soap:operation xmlns:soap="http://schemas.xmlsoap.org/soap/" soapAction="create" style="document"/>
90     <wsdl:input xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name="create">
91       <soap:body xmlns:soap="http://schemas.xmlsoap.org/soap/" use="literal"/>
92     </wsdl:input>
93     <wsdl:output xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name="createResponse">
94       <soap:body xmlns:soap="http://schemas.xmlsoap.org/soap/" use="literal"/>
95     </wsdl:output>
96   </wsdl:operation>
97   <wsdl:operation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name="querySPARQL">
98     <soap:operation xmlns:soap="http://schemas.xmlsoap.org/soap/" soapAction="querySPARQL" style="document"/>
99     <wsdl:input xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name="querySPARQL">
100       <soap:body xmlns:soap="http://schemas.xmlsoap.org/soap/" use="literal"/>
101     </wsdl:input>
102     <wsdl:output xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name="querySPARQLResponse">
103       <soap:body xmlns:soap="http://schemas.xmlsoap.org/soap/" use="literal"/>
104     </wsdl:output>
105   </wsdl:operation>
106   <wsdl:operation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name="plug">
107     <soap:operation xmlns:soap="http://schemas.xmlsoap.org/soap/" soapAction="plug" style="document"/>
108     <wsdl:input xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name="plug">
109       <soap:body xmlns:soap="http://schemas.xmlsoap.org/soap/" use="literal"/>
110     </wsdl:input>
111     <wsdl:output xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name="plugResponse">
112       <soap:body xmlns:soap="http://schemas.xmlsoap.org/soap/" use="literal"/>
113     </wsdl:output>
114   </wsdl:operation>
115 </wsdl:binding>
116 <wsdl:service xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name="WebService">
117   <wsdl:port xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" binding="tns:WebServiceSoapBinding" name="WebServicePort">
118     <soap:address xmlns:soap="http://schemas.xmlsoap.org/soap/" location="http://localhost:8080/WebService/">
119   </wsdl:port>
120 </wsdl:service>
121 </wsdl:definitions>
122

```

Figure 4 WSDL service for Public API of the Rules Inference Engine

The second API made available by the Rules Inference Engine concerns the *Evaluator API*. This API allows the internal integration of evaluators for different technologies into the Rules Inference Engine, as already explained in the Architecture Section of this report.

The Evaluator API has been defined to cope with the *execution algorithm* presented in the previous Section. To take advantage of this feature, in order to connect an



evaluator to the Rules Inference Engine, it must adhere to the following API conventions:

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface BindNamespace {
    String namespace();
}

@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface BindTest {
    String namespace() default "";
    String endpoint();
}
```

**Figure 5 Bindings for Namespaces and End Points**

Typically, test criteria belonging to the same domain-specific set of concepts (i.e., same accessibility guideline) share a common namespace within their enclosing ontology. To distinguish between each criterion, a unique identifier must be present, i.e., specified as a URI in the domain-specific ontology. The identification of each criterion is, then, the conjunction of a namespace and an end point identifying the criterion within this namespace (e.g., <http://example.com/ex.owl#Test01>), as suggested in OWL ontology engineering practices.

This type of assertions supports the application of different analysis schemes to a given criterion, since each criterion is uniquely identifiable by a given URI, e.g., <http://www.w3.org/TR/WCAG20/#text-equiv-all> for WCAG 2.0 criterion 1.1.1. The concretization of such analysis schemes is implementation-defined through correspondent techniques for each technology. For instance, for criterion 1.1.1, WCAG 2.0 defines a set of techniques that cover HTML, CSS, and Flash (c.f. <http://www.w3.org/WAI/WCAG20/quickref/#qr-text-equiv-all>) technologies. Still, the URL referring to the accessibility barrier remains the same (i.e., criterion 1.1.1).

Hence, the notions of *namespace* and *end point* can be directly mapped into *class* and *method*, correspondingly. The way these mappings have been specified is depicted above in Figure 5. In the first case, we have defined a class-level annotation (*ElementType.TYPE*) which specifies the overall namespace aggregating a set of criterion-based software methods. In the latter case, criteria are specified with method-level annotations (*ElementType.METHOD*) encompassing an optional namespace (for those cases where a criterion is specified in a different namespace within its corresponding domain-specific ontology).

It is worth noticing that both annotation specifications have been marked with a *RetentionPolicy.RUNTIME* meta-annotation. This is a specific detail required by the JVM reflection API, in order to have these annotations available at execution time (which can be leveraged, e.g., by evaluators).

Next, in Figure 6 we present a simple example on how these annotations can be used in Java. Here, we have defined a class, *ExampleModule*, which is responsible for

implementing the guidelines specified in *http://example.com/ex.owl*. Within this class we have declared two methods, *Test01* and *Test02* which implement two different tests specified in the guideline with *http://example.com/ex.owl#Test\_01* and *http://example.com/ex.owl#Test\_02* URIs, correspondingly. A third method, *Test03*, exemplifies how to implement a test criterion for the same guideline but residing on a different namespace, e.g., specified by the *http://example.com/ex2.owl#Test\_03* URI:

```
@BindNamespace(namespace="http://example.com/ex.owl")
public class ExampleModule {

    @BindTest(endpoint="#Test_01")
    public Object Test01(Object app) {
        // actual evaluation code goes here
    }

    @BindTest(endpoint="#Test_02")
    public Object Test02(Object app) {
        // actual evaluation code goes here
    }

    @BindTest(
        namespace="http://example.com/ex2.owl",
        endpoint="#Test_03"
    )
    public Object Test03(Object app) {
        // actual evaluation code goes here
    }
}
```

**Figure 6 Integrating a set of bindings in a software component**

## 3 Rules User Interface

This chapter describes the ACCESSIBLE Rules User Interface. The intention of the Rules User Interface is to support users without deeper knowledge about the concept of ontologies enhancing the ACCESSIBLE ontological knowledge base with new accessibility standards, guidelines and assessment rules.

We start with an introduction examining the aim of the tool in more detail and how it is integrated in the ACCESSIBLE system architecture.

### 3.1 Introduction

As has been already pointed out in chapter 2.1 accessibility is not a uniform concept. What is regarded as accessible varies in the first case dependent on the user requirements including the disabilities and impairments as well as the skills and capabilities of the addressed users. But accessibility aspects also differ to a considerable extent relative to the application area and the used assistive technologies. And last but not least which of the diversity of accessibility concepts are assumed to be the most suitable for the own requirements is also a question of personal conviction.

For that reason it is important that people can find the accessibility aspects relevant for their specific needs within the plurality of existing accessible concepts. Furthermore they should be able to create new own accessibility concepts adapted to their individual demands out of these aspects.

In the ACCESSIBLE ontology based knowledge resource all currently available important accessibility standards for different application areas and user characteristics are gathered and combined to a holistic approach. The ACCESSIBLE Rules User Interface allows the user to search within this collection of accessibility concepts and to make new combinations of different accessibility aspects that meet best one's particular requirements.

Another aspect that should be considered in this context is the ongoing progression of the emerging area of accessibility. It is an aim of the ACCESSIBLE project to raise the awareness of the necessity of accessibility and hence to push on its development. So we expect further advancements of that field in the future and we wish to have the results of that process also covered in the project. That means that the extensibility of the ACCESSIBLE ontological knowledge base is essential for the success of the project.

One step to obtain scalability of the ACCESSIBLE ontology based knowledge resource is to distinguish general and domain specific concepts in the architecture of the ontological knowledge base as described in D4.1. The immutable general concepts determine the structure of the architecture and thus form the basis for the communication with different components of the ACCESSIBLE system architecture, whereas an arbitrary number of new specific concepts can be added at any time to the ACCESSIBLE ontological knowledge base.

The consequent step is to provide methods for integrating new accessibility concepts into the ACCESSIBLE ontological knowledge base. People experienced in the area of ontologies can use existing ontology management tools (like Protégé [2]) for editing the ACCESSIBLE ontology based knowledge resource. But for users who do not have the necessary know-how an alternative way has to be offered. The ACCESSIBLE Rules User Interface is such an alternative to conventional ontology management tools. It enables the user to enhance the ACCESSIBLE knowledge base with new accessibility standards, guidelines and assessment rules in a simple and intuitive way without the need for any ontology-specific knowledge.

To achieve the goals of the ACCESSIBLE Rules User Interface presented above, some specific requirements have been defined which will be introduced in the next section.

### **3.2 Requirements**

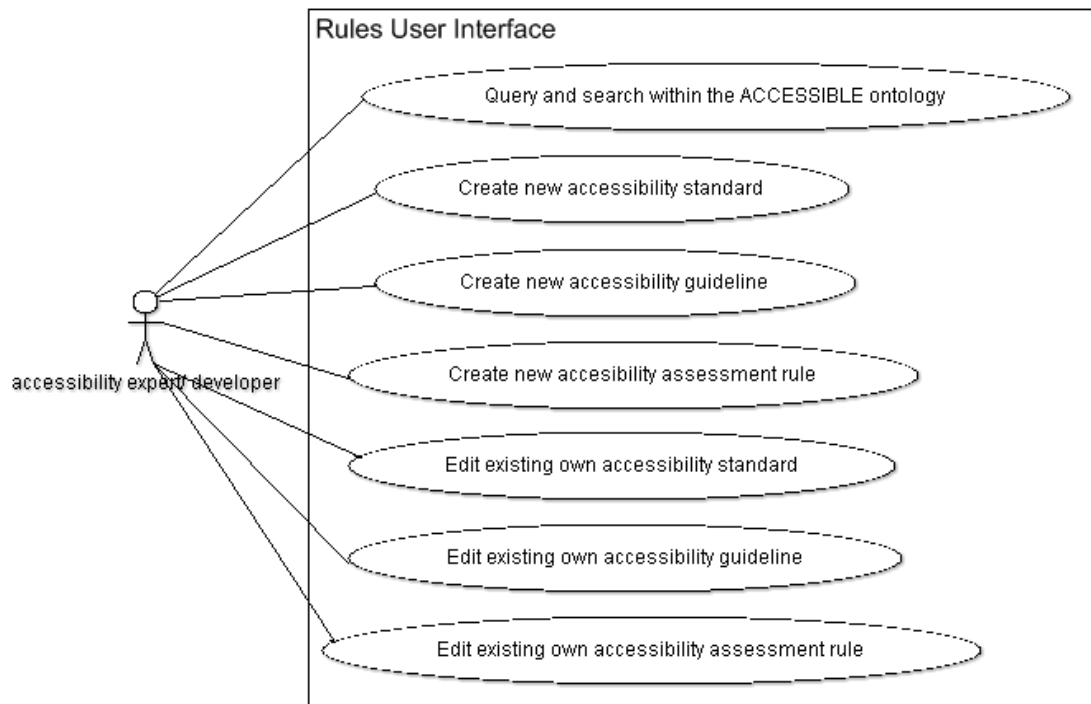
As described in the previous section the main purpose of the ACCESSIBLE Rules User Interface is to provide a possibility to search within the accessible based knowledge resource and to integrate new accessibility concepts. That may be motivated either by the demand for making the ontological knowledge base fit personal requirements or for updating the ontology with the latest developments in the area of accessibility.

Accessibility concepts that can be created with the ACCESSIBLE Rules User Interface comprise accessibility standards, guidelines and assessment rules. Accessibility assessment rules describe which checkpoints should be used for specific user and application requirements. Accessibility guidelines are combinations of different accessibility assessment rules and can be grouped in accessibility standards.

Because the development of accessibility concepts is an ongoing process it is also essential to have the possibility to modify once created accessibility concepts. Therefore the Rules User Interface has to support the editing of own accessibility assessment rules, guidelines and standards.

Another function of the ACCESSIBLE Rules User Interface is to enable the user to query and search within the ACCESSIBLE ontological knowledge base. This functionality is in first case needed to support the user in finding the data required to create new accessibility concepts. But it is also possible that the querying and searching is done separately and independent of the process of editing the ACCESSIBLE knowledge base.

All functional requirements are depicted in the Use Case diagram in Figure 7.



**Figure 7 Use Case diagram of the ACCESSIBLE Rules User Interface**

From the definition of the target user group of the ACCESSIBLE Rules User Interface as people without deeper knowledge about ontologies we derived the following design requirement: The user should be able to perform all supported tasks, determined by the described functional requirements, without reading or writing any ontology related syntax. This is very important, as it is a distinctive feature of the Rules User Interface towards existing ontology management tools.

In the next paragraph the technical requirements for the ACCESSIBLE Rules User Interface are presented.

The new accessibility concepts created with the Rules User Interface are integrated in the ACCESSIBLE ontological knowledge base. Therefore the same format as in the ACCESSIBLE ontology based architecture must be used for new accessibility concepts. That means that accessibility standards have to be saved as OWL files and accessibility guidelines as OWL classes of the OWL file representing the corresponding accessibility standard. Accessibility assessment rules have to be translated into SWRL rules and incorporated into the OWL file of the belonging accessibility standard.

For querying and searching within the ACCESSIBLE ontological knowledge base, SPARQL should be used. This is necessary to be able to cooperate with the ACCESSIBLE Rules Inference Engine.

Besides the functional, technical and the design requirements there are also requirements regarding the right management:

With the ACCESSIBLE ontology based knowledge resource we want to provide a reliable source of proper accessibility related information. To ensure that the gathered data remains clean we have to protect the ACCESSIBLE knowledge resource towards unwanted modifications. Because of that the changes possible with the Rules User Interface are only performed on a local copy of the ACCESSIBLE ontology based

knowledge resource. It is not possible to upload any adjustments to the official ACCESSIBLE ontological knowledge resource, which is available online, without the permission of a member of the ACCESSIBLE maintenance team.

So with the Rules User Interface any interested user is able to adapt his own version of ACCESSIBLE ontological knowledge resource to his personal requirements. However, the publish of the updated ontologies in the ACCESSIBLE portal can be permitted only from authorised representatives that are responsible for the ontologies maintenance, If the review of the suggested enhancement is positive from the authorised user then the provided changes can be adopted for the official ACCESSIBLE ontological knowledge base.

How the defined requirements are realised in the Rules User Interface will be presented in the next chapter.

### 3.3 The Rules User Interface

The ACCESSIBLE Rules User Interface consists of three main components: A Graphical User Interface, the Ontology Management Module and a Search Module.

The architecture of the ACCESSIBLE Rules User Interface as well as the relationships to other components of the ACCESSIBLE system architecture is shown in Figure 8.

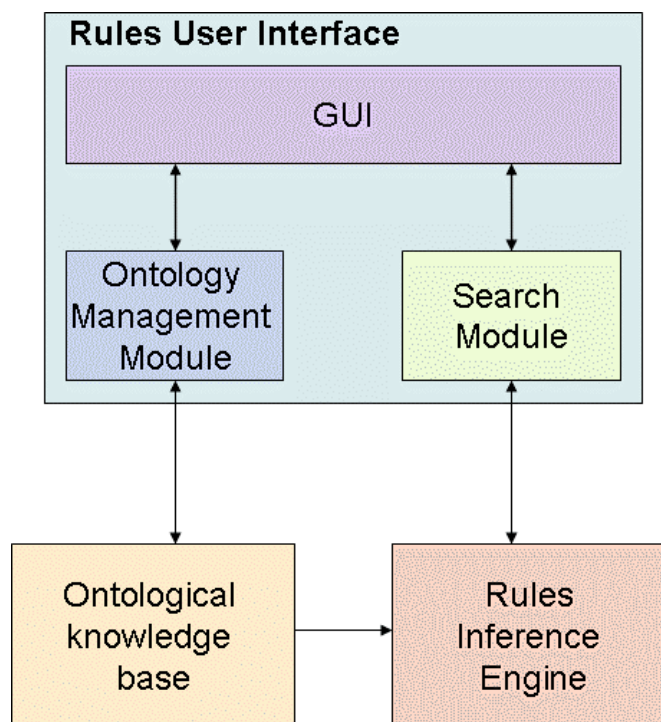


Figure 8 Architecture of the ACCESSIBLE Rules User Interface

All components are described now in detail.

### 3.3.1 The Graphical User Interface

The graphical user interface obtains data from the ACCESSIBLE Ontology and from the Rules Inference Engine to provide the user with the information necessary to perform the tasks supported by the ACCESSIBLE Rules User Interface.

The start screen of the tool depicted in Figure 9 shows an overview of all accessibility concepts the user has already created with the Rules User Interface. They are arranged in a tree structure to reveal the coherences between them. On the highest level all own accessibility standards are listed. Each standard contains all belonging own guidelines and each guideline contains all belonging own accessibility assessment rules.

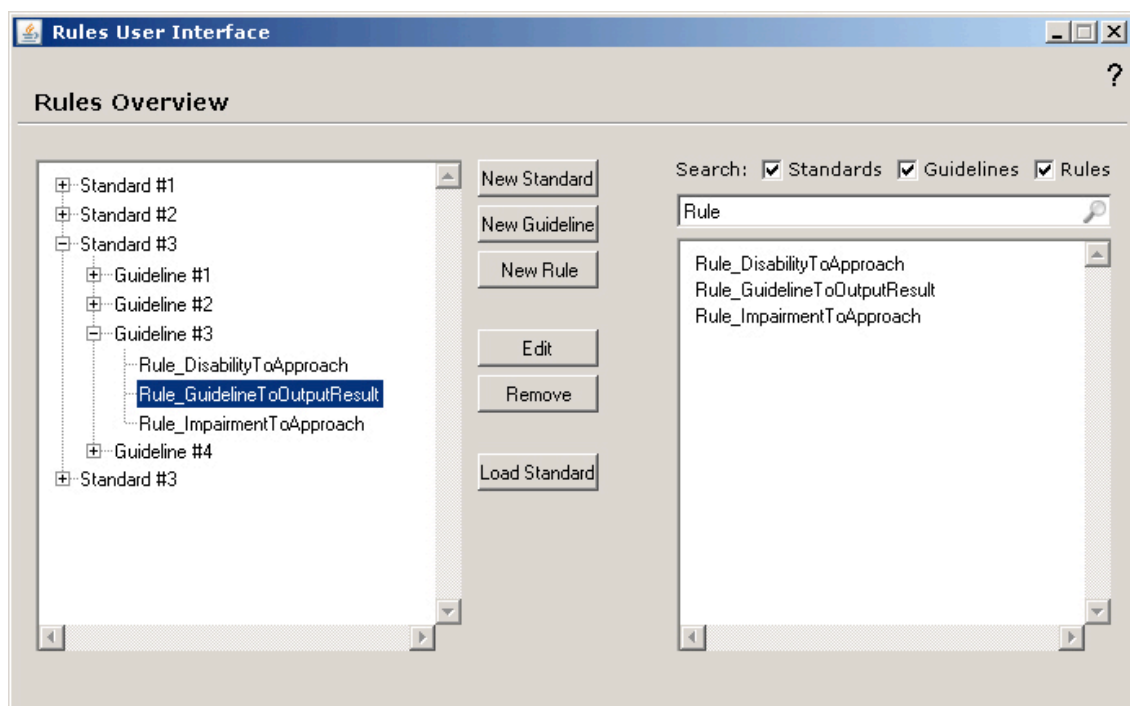


Figure 9 Start screen of the Rules User Interface

To help the user finding the required concepts in the overview there is a special search functionality offered on the same screen. With the search mask one can determine which kind of concepts s/he is looking for: own standards and/or own guidelines and/or own assessment rules. The search results will contain all selected concepts matching the given search word and will be listed on the screen. When selecting an entry on the list of search results the corresponding element in the overview will be highlighted in order to enable the user to work with it.

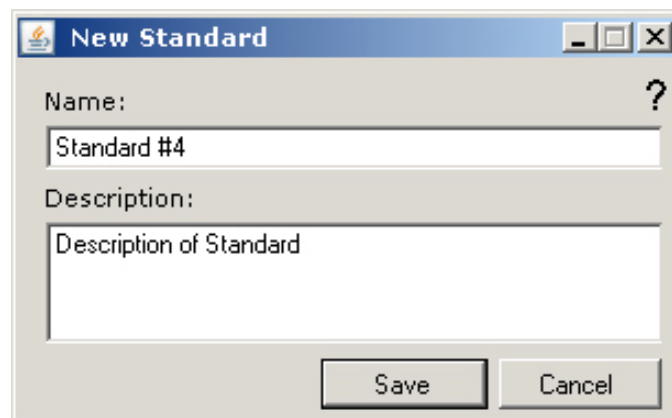
That screen is the starting point for all defined functional requirements of the ACCESSIBLE Rules User Interface:

To view and edit own accessibility concepts the element representing the specific concept in the overview has to be selected and the button “edit” has to be pressed. That will open a new window presenting the selected component in detail and providing the possibility to make changes.

All own concepts listed in the overview can also be removed, by selecting the corresponding element in the tree and pressing the “delete” button. If a standard is deleted all belonging guidelines will also be removed. Similarly all belonging assessment rules are deleted when deleting a guideline. The user is informed about that and asked for a confirmation before the concepts are indefinitely deleted.

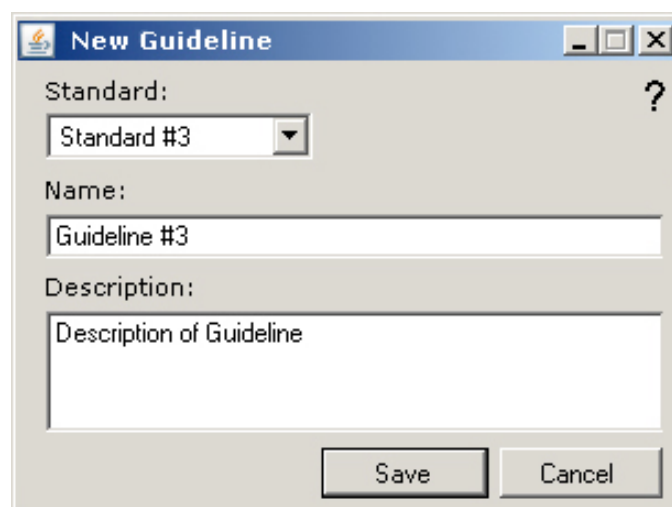
To create new concepts the corresponding buttons “new standard”, “new guideline” and “new rule” can be used for performing the task. If a new guideline is specified the user has the possibility to select an accessibility standard in the overview to which the new guideline should belong to before pressing the button “new guideline”. Likewise when creating a new rule the guideline to which the rule should belong to can be determined by selecting the corresponding element in the overview before pressing the button “new rule”.

The screens for creating a new standard, guideline or rule are very similar to the screens for editing them. The only difference is that in the screens for editing existing concepts the graphical elements are already predefined with the attributes of the represented concepts.



**Figure 10** Screen for creating a new accessibility standard

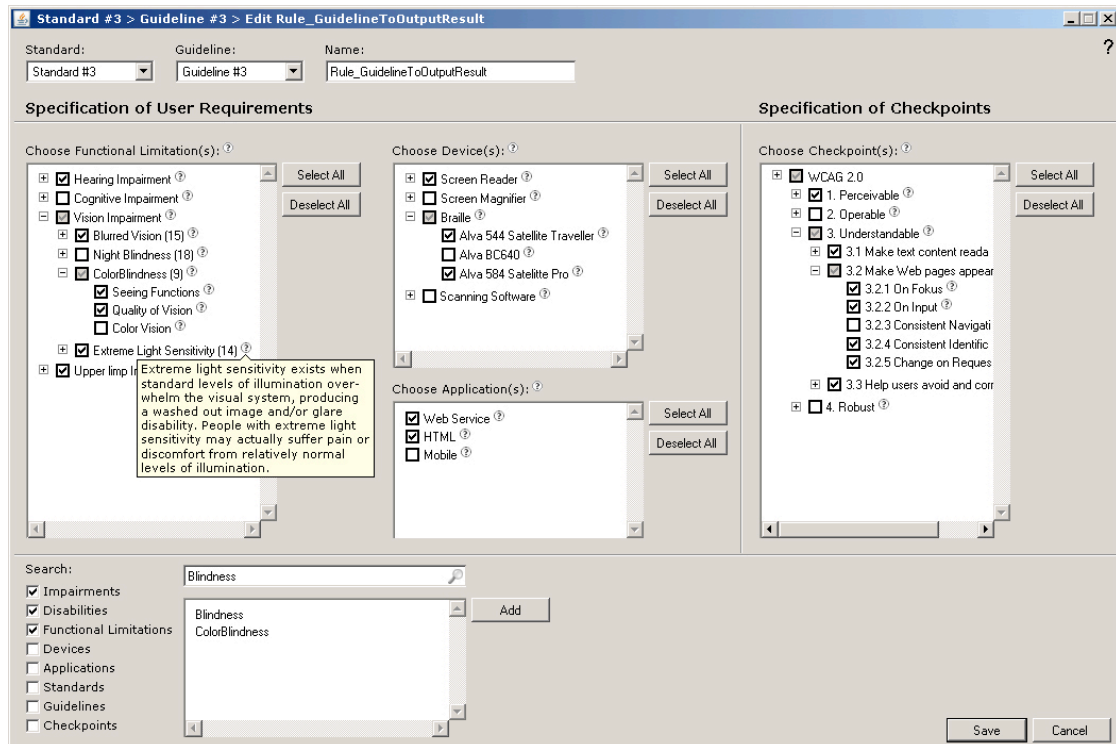
Figure 10 shows the screen representing an accessibility standard with a field for the name of the standard and another one to describe the standard in some words.



**Figure 11** Screen for creating a new accessibility guideline



As can be seen in Figure 11 the screen for editing or creating an accessibility guideline looks almost the same as the one for accessibility standards. A guideline also has a name and a short description. In addition it has to be determined to which standard the guideline belongs to.



**Figure 12** Screen for specifying or editing an own accessibility assessment rule

A more complex screen is necessary for specifying a new or editing an existing accessibility assessment rule. It is depicted in Figure 12. On top of the screen the standard and the guideline to which the rule belongs to and the name of the rule are defined.

On the left side is the area for specifying the user and application requirements. That implies the selection of functional limitations, devices and applications that should be considered in the assessment rule. There is a list with all devices covered in the ACCESSIBLE ontological knowledge resource, a list of all applications and a tree with all impairments in the highest level, matching disabilities in the next level and associated functional limitations in the lowest level. The user has the possibility to select either a single functional limitation or all functional limitations of a disability or all functional limitations of all disabilities of a single impairment. The tree structure also makes it easy to find the required functional limitations.

On the right side the checkpoints of the rule are selected. Checkpoints are listed in a tree structure with accessibility standards on the highest level. Accessibility standards do contain accessibility guidelines and accessibility standards contain checkpoints. This arrangement again first helps the user to find the required checkpoints and secondly enables the user to select not only single checkpoints but also all checkpoints of a guideline or all checkpoints of all guidelines of an accessibility standard with one single click.

On the lower area there is a search mask for querying the ACCESSIBLE ontology based knowledge resource. With the checkboxes one can determine which categories should be searched for. As a result all elements of the selected category matching the search word will be listed. With the button on the right an entry of the list of search results can directly be added to the current selection for the specification of the assessment rule.

The description of the graphical user interface of the ACCESSIBLE Rules User Interface shows that the design requirement of having a GUI free from programming code and ontology related syntax is complied with.

### **3.3.2 The Ontology Management Module**

The Ontology Management Module is responsible for the enhancement of the ACCESSIBLE ontology based knowledge resource with new accessibility concepts. The process of adding new accessibility concepts to the knowledge base is comprised of the following steps.

First the new component has to be created. Accessibility standards are instantiated as OWL files, accessibility guidelines as OWL classes and assessment rules as SWRL rules.

The new concepts then have to be connected with each other. That means that SWRL rules have to be inserted in the OWL file of the accessibility standard to which the accessibility assessment rule should belong to. Accordingly OWL classes have to be part of the OWL file of the accessibility standard which the accessibility guideline should appertain to.

In a last step the extensions are integrated in the ACCESSIBLE Ontology by creating the essential bindings between existing and new concepts. To be more precise a mapping between the new domain specific OWL files and the generic concepts of the ACCESSIBLE ontology architecture has to be done.

Besides the construction and integration of new accessibility concepts also the editing of concepts and rules of the ACCESSIBLE Ontology is done by the Ontology Management Module by altering the corresponding OWL files, OWL classes and SWRL rules.

### **3.3.3 The Search Module**

The Search Module of the Rules User Interface enables the user to query and search within the ACCESSIBLE ontology based knowledge resource. The graphical user interface provides a search mask for creating queries for specific data (e.g. all guidelines matching a specific search word). The queries are translated into SPARQL queries by the Search Module and sent to the ACCESSIBLE Rules Inference Engine for execution. The subset of accessibility concepts of the ACCESSIBLE ontology

matching the query are then delivered back to the Search Module and the search results are presented to the user through the graphical user interface.

### 3.4 Implementation

The following software components, libraries, standards and languages were used for the implementation of the ACCESSIBLE Rules User Interface:

- 1) *JVM (Java Virtual Machine) technologies and libraries*: the implementation of the Rules User Interface is based on the JVM and open source Java libraries. For the implementation of the graphical user interface the Java Swing library was used.
- 2) *OWL API<sup>5</sup>*: The OWL API is an open source Java API for creating, editing and serialising OWL ontologies. We decided to use the OWL API for working with the ACCESSIBLE ontological knowledge resource in the Rules User Interface because of the limited but sufficient functionality and the detailed documentation available.
- 3) *SWRL<sup>6</sup> (Semantic Web Rule Language)*: SWRL is a W3C standard for defining Horn-like rules using the Web Ontology Language (OWL). The accessibility assessment rules are realized as SWRL rules within the ACCESSIBLE Rules User Interface and the ACCESSIBLE ontological knowledge base.
- 4) *SPARQL<sup>7</sup> (Query Language for RDF)*: SPARQL is a query language standardized by W3C for expressing queries across diverse data sources. For querying and searching within the ACCESSIBLE ontological knowledge base with the Rules User Interface, SPARQL queries are used to receive the required data from the ACCESSIBLE Rules Inference Engine.

A very important part of the process of creating new accessibility concepts with the ACCESSIBLE Rules User Interface is the transformation of accessibility assessment rules into SWRL rules. This procedure is illustrated by an example accessibility assessment rule in Figure 13.

---

<sup>5</sup> <http://owlapi.sourceforge.net/index.html>

<sup>6</sup> <http://www.w3.org/Submission/SWRL/>

<sup>7</sup> <http://www.w3.org/TR/rdf-sparql-query>

```
FunctionalLimitation_belongsTo_User(low_vision, ?a) ∧  
Device_linksTo_User(screenreader, ?a)
```



```
User_linksTo_Checkpoint(?a, WCAG2_cp2.1.1) ∧  
User_linksTo_Checkpoint(?a, WCAG2_cp2.2.1)
```

**Figure 13 Example accessibility assessment rule in SWRL format**

The information about the specified application, device and user characteristics is covered by the antecedent of the SWRL rule. A fictive user is assigned with the requirements that should be considered in the accessibility assessment rule by using existing properties of the ACCESSIBLE Generic Ontology. For more details about the Generic Ontology and its properties see D4.1. In the example rule the user has the functional limitation “low vision” and uses a “screen reader” as assistive device.

The consequent of the SWRL rule contains the information about the specified checkpoints that should be applied within the accessibility assessment rule. The checkpoints are linked with the same fictive user as used in the antecedent by using existing properties of the ACCESSIBLE Generic Ontology. This mapping is important for binding the user requirements of the antecedent to the set of checkpoints of the consequent of the rule.

In the example case the “checkpoints 2.1.1 and 2.2.1” of the web accessibility standard “WCAG 2.0” is selected. So the example accessibility assessment rule in a whole says that for low vision users using a screen reader the checkpoints 2.1.1 and 2.2.1 of WCAG 2.0 should be applied.

## 4 Conclusions

This deliverable presented the core software components of ACCESSIBLE's Services and Content Knowledge Infrastructure: the Rules Inference Engine, and the Rules User Interface.

Through these components, the ACCESSIBLE project provides a way to infer accessibility knowledge (e.g., which accessibility tests are appropriate for a given disability), which can be used by accessibility evaluators – such as those being developed in WP5 –, in order to deliver semantic accessibility assessment.

The improvement of these components is being expanded to Task 4.4 (Testing and Validation), where all WP4 components are being tested, validated, in order to be made available as standalone products to be integrated into any software packages (e.g., non-ACCESSIBLE accessibility evaluators).

## References

Since the Rules Inference Engine by itself has little to no use, its integration in other technologies, products, and services is crucial. For this to successfully happen, the previous requirement must be accomplished. However, it is also important to provide proper documentation (technical, tutorials, guides, etc.) on how to integrate the Rules Inference Engine outside the scope of the ACCESSIBLE project. The integration of this component in other ACCESSIBLE components (such as the accessibility evaluation tools) provides a first solid ground on this issue. While this deliverable (D4.2) details the implementation aspects of the Rules Inference Engine, these aspects will be further detailed in the deliverable D4.3 [1] D4.3 “A set of guidelines for the validation and integration of the implemented tools and methodologies”

The consequent step is to provide methods for integrating new accessibility concepts into the ACCESSIBLE ontological knowledge base. People experienced in the area of ontologies can use existing ontology management tools (like Protégé [2] The Protégé Ontology Editor and Knowledge Acquisition System –at: <http://protege.stanford.edu/>