Project:
Accessibility Assessment Simulation Environment for New Applications Design and Development
_ACCESSIBLE_, Grant Agreement No. 224145)

Deliverable number and title:
D 3.3a – ACCESSIBLE system architecture specification (beta)

<table>
<thead>
<tr>
<th>Lead beneficiary:</th>
<th>CERTH/ITI</th>
</tr>
</thead>
<tbody>
<tr>
<td>WP. no, title and activity type</td>
<td>WP3 – Design Infrastructure and ACCESSIBLE harmonized Methodology</td>
</tr>
<tr>
<td>Contributing Task(s)</td>
<td>T2.1, T2.2, T2.3, T2.4</td>
</tr>
<tr>
<td>Dissemination level</td>
<td>RE</td>
</tr>
<tr>
<td>Delivery date</td>
<td>Month 12</td>
</tr>
<tr>
<td>Status</td>
<td>Final Draft</td>
</tr>
<tr>
<td>File name and size</td>
<td>ACCESSIBLE-ITI-WP3-R-V13-Del3.3a.doc,</td>
</tr>
</tbody>
</table>
## Authors List

### Leading Author (Editor)

<table>
<thead>
<tr>
<th>Surname</th>
<th>Initials</th>
<th>Beneficiary Name (Short Name)</th>
<th>Contact email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tzovaras</td>
<td>D.</td>
<td>CERTH/ITI</td>
<td><a href="mailto:Dimitrios.tzovaras@iti.gr">Dimitrios.tzovaras@iti.gr</a></td>
</tr>
</tbody>
</table>

### Co-authors (In alphabetic order)

<table>
<thead>
<tr>
<th>#</th>
<th>Surname</th>
<th>Initials</th>
<th>Beneficiary Name (Short Name)</th>
<th>Contact email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Votis</td>
<td>K.</td>
<td>CERTH/ITI</td>
<td><a href="mailto:kvotis@iti.gr">kvotis@iti.gr</a></td>
</tr>
<tr>
<td>2</td>
<td>Kaklanis</td>
<td>N.</td>
<td>CERTH/ITI</td>
<td><a href="mailto:nkak@iti.gr">nkak@iti.gr</a></td>
</tr>
<tr>
<td>3</td>
<td>Oikonomou</td>
<td>T.</td>
<td>CERTH/ITI</td>
<td><a href="mailto:tholikon@iti.gr">tholikon@iti.gr</a></td>
</tr>
<tr>
<td>4</td>
<td>Kastori</td>
<td>M.</td>
<td>CERTH/ITI</td>
<td><a href="mailto:gkastori@iti.gr">gkastori@iti.gr</a></td>
</tr>
<tr>
<td>5</td>
<td>Lopes</td>
<td>R.</td>
<td>FFCUL</td>
<td><a href="mailto:rlopes@di.fc.ul.pt">rlopes@di.fc.ul.pt</a></td>
</tr>
<tr>
<td>6</td>
<td>Michael Loupis</td>
<td>C.</td>
<td>SOLINET</td>
<td><a href="mailto:M.Loupis@SOLINET.com">M.Loupis@SOLINET.com</a></td>
</tr>
<tr>
<td>7</td>
<td>Ringler</td>
<td>M.</td>
<td>NETSCOUTS</td>
<td><a href="mailto:ringler@netscouts-gmbh.de">ringler@netscouts-gmbh.de</a></td>
</tr>
<tr>
<td>8</td>
<td>Papadopoulou</td>
<td>M.</td>
<td>FORTH</td>
<td><a href="mailto:mpapadop@ics.forth.gr">mpapadop@ics.forth.gr</a></td>
</tr>
<tr>
<td>9</td>
<td>Partarakis</td>
<td>N.</td>
<td>FORTH</td>
<td><a href="mailto:partarak@ics.forth.gr">partarak@ics.forth.gr</a></td>
</tr>
<tr>
<td>10</td>
<td>Korn</td>
<td>P.</td>
<td>SUN</td>
<td><a href="mailto:Peter.Korn@sun.com">Peter.Korn@sun.com</a></td>
</tr>
</tbody>
</table>

## Peer Reviewers List

<table>
<thead>
<tr>
<th>#</th>
<th>Surname</th>
<th>Initials</th>
<th>Contact email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bandeira</td>
<td>R.</td>
<td><a href="mailto:rbandeira@lasige.di.fc.ul.pt">rbandeira@lasige.di.fc.ul.pt</a></td>
</tr>
<tr>
<td>2</td>
<td>Kavadias</td>
<td>C.</td>
<td><a href="mailto:C.Kavadias@solinet.com">C.Kavadias@solinet.com</a></td>
</tr>
</tbody>
</table>
Executive Summary

This deliverable presents the initial system architecture of ACCESSIBLE. The ACCESSIBLE project aims to research and develop an Open Source Assessment Simulation and accreditation environment, for collating and merging different methodological tools, checking the coherence with the W3C/WAI ARIA and other standardisation works in order to enable large organisations, SMEs or individuals (developers, designers, etc.) to produce software products of superior accessibility and quality, accompanied with appropriate measures, technologies and tools that improve their accessibility.

The document which is the output of work package 3, it covers the development of the main concepts of the overall ACCESSIBLE architecture. Thus, it describes the beta version of the system architecture along with the technologies that can serve the project requirements as defined within the T2.2. It is based on the results of tasks 2.3 (Accessibility, assessment simulation and system requirements), 2.1 (User Needs) and 2.2 (Literature review and market survey), T2.4 “Use cases and application scenarios”, and will provides a detailed and closed specification of the overall system design methodology and all design steps and models/systems involved, the overall system components, its behavior, and its interactions. Furthermore, this report presents analytically the architecture of each module of the ACCESSIBLE platform. This architecture focuses on the conceptual and physical implementation of the software components as well as their intra-communication architecture.

It should be noted that although this document “ACCESSIBLE system architecture specification (beta)” is delivered according to the ACCESSIBLE plan in Month 12, the system architecture will remain an open issue till all its subcomponents and subsystems are built and an updated version of this document will be provided at month 18 through the deliverable D3.3b: ACCESSIBLE system architecture specification (final)
Table of contents

Authors List .............................................................................................................. 3
Peer Reviewers List ................................................................................................. 3
Executive Summary ................................................................................................. 4
Table of contents ..................................................................................................... 5
List of figures ........................................................................................................... 6
List of tables ............................................................................................................ 7
List of abbreviations and acronyms ....................................................................... 9

1 Introduction ........................................................................................................... 10
   1.1 Scope .............................................................................................................. 10

2 Methodology ......................................................................................................... 12
   2.1 Objective and Definition of System Architecture .............................................. 12
   2.2 Architecture development process .................................................................. 14
   2.3 Process for the development of the ACCESSIBLE System ............................. 15
       2.3.1 General Description .................................................................................. 15
       2.3.2 Architecture Planning within ACCESSIBLE project ............................... 17
   2.4 Architectural description in conformance with IEEE standards ..................... 18
   2.5 System Concept ............................................................................................. 20
   2.6 Impact of Requirements to the ACCESSIBLE Architecture ......................... 21
       2.6.1 ACCESSIBLE Generic Requirements ..................................................... 22
       2.6.2 ACCESSIBLE Assessment Simulation Module Requirements ................. 27
       2.6.3 ACCESSIBLE ontological Knowledge Resource Requirements ............... 35
       2.6.4 User Interface Portal Requirements ......................................................... 37
       2.6.5 Stand alone interface requirements .......................................................... 41
       2.6.6 ACCESSIBLE Developer/designer–aid Module Requirements ................. 45
       2.6.7 ACCESSIBLE EARL-based reporting tool Requirements ......................... 48

3 ACCESSIBLE Conceptual Architecture ............................................................... 51
   3.1 ACCESSIBLE Architectural scheme ............................................................... 51
       3.1.1 ACCESSIBLE User Interface .................................................................. 52
       3.1.2 ACCESSIBLE Assessment Simulation Modules ........................................ 53
       3.1.3 ACCESSIBLE ontological knowledge Resource ...................................... 54
       3.1.4 ACCESSIBLE Developer and designer aid module .................................. 55
       3.1.5 ACCESSIBLE Generated Analysis Reporting Tool .................................. 55
   3.2 ACCESSIBLE subsystems Logical architecture ............................................. 56
       3.2.1 ACCESSIBLE Assessment Simulation Module ........................................ 56
       3.2.2 ACCESSIBLE ontological Knowledge Resource ...................................... 71
       3.2.3 ACCESSIBLE Portal/ User Interface Architecture .................................... 74
       3.2.4 ACCESSIBLE Developer/designer–aid module ........................................ 82
       3.2.5 The EARL-based reporting tool ............................................................... 84

4 Conclusions and next steps .................................................................................. 87

5 Annex .................................................................................................................. 88
   5.1 SAFIRE .......................................................................................................... 88
   5.2 ASK-IT Alignment tool ................................................................................... 90
   5.3 SET Tool ......................................................................................................... 92

References ............................................................................................................... 93
List of figures
Figure 1 Model of the ACCESSIBLE System architecture ........................................ 12
Figure 2 Procedure to derive the ACCESSIBLE System architecture ............................. 14
Figure 3: Aside Deficiencies in Traditional Lifecycle Models ........................................... 15
Figure 4: Waterfall “Ultimate Delivery” Syndrome ...................................................... 17
Figure 5: Massive Parallelism” Syndrome ..................................................................... 17
Figure 6: Incremental and iterative process within ACCESSIBLE ...................................... 18
Figure 7: Illustration of the conceptual framework of IEEE 1471 standard for Architecture Description ......................................................................................................... 19
Figure 8: ACCESSIBLE conceptual functionalities ......................................................... 21
Figure 9. Overall Architecture of ACCESSIBLE system ................................................ 52
Figure 10: Logical architectural scheme of the Web Applications Assessment standalone tool ....................................................................................................................... 58
Figure 11: Logical architectural scheme of the Web Applications Assessment tool (Web services version) .................................................................................................. 61
Figure 12. Mobile Web Accessibility Evaluator Architecture .......................................... 63
Figure 13: Block Diagram of the Accessible Web Service Assessment Module ............... 65
Figure 14: Overview of the Accessible WS Evaluation process ....................................... 66
Figure 15: SDL System- GUI application flowchart diagram ........................................... 68
Figure 16: System Architecture of the Accessibility Assessment Module for SDL Systems .......................................................................................................................... 70
Figure 17. Example of the assessment report text file ..................................................... 71
Figure 18: Accessible Ontologies Architecture ................................................................... 73
Figure 19. Rules Inference Engine Architecture ............................................................. 74
Figure 20. Extending the ACCESSIBLE architecture with the building blocks of the user interface ................................................................................................................. 75
Figure 21. ACCESSIBLE User Portal architecture ......................................................... 76
Figure 22: The overall process flow in the data access layer ............................................. 77
Figure 23: The abstract structure of the XML descriptor file .......................................... 77
Figure 24: The result of a web-service invocation ......................................................... 78
Figure 25: Architecture of Unified Web-based Interfaces (UWIs) as employed by EAGER .................................................................................................................................... 79
Figure 26: Architecture of ACCESSIBLE stand alone tool User Interface ................. 81
Figure 27: An example of an ACCESSIBLE xml based project specification .................. 81
Figure 28: Logical architecture of the ACCESSIBLE Developer/designer-aid Module ............................................................................................................................... 83
Figure 29: Overview of the Simulation process ........................................................ ...... 84
Figure 30 Accessible EARL-based Reporting Tool Overall Architecture ...................... 85
Figure 31: SAFIRE Editor sample .................................................................................. 89
Figure 32: SAFIRE Editor Sample .................................................................................. 90
Figure 33 CERTH-ITI’s ASK-IT Service Alignment Tool’s service models ..................... 91
Figure 34 Alignment of a service .................................................................................... 91
Figure 35. SET tool ....................................................................................................... 92
List of tables

Table 1: Principles to approach development process ................................................................. 16
Table 2: Lifecycle Risks and Actions ............................................................................................... 18
Table 3: SDL source code sample .................................................................................................... 68
## List of abbreviations and acronyms

*(in alphabetic order)*

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>EARL</td>
<td>Evaluation and Report Language</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standards Institute</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ICF</td>
<td>International Classification of Functioning, Disability and Health</td>
</tr>
<tr>
<td>ICT</td>
<td>Information and communication technologies</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>ITU-T</td>
<td>Telecommunication Standardization Sector</td>
</tr>
<tr>
<td>JAAPI</td>
<td>Java Accessibility API</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDF-S</td>
<td>RDF-Schema</td>
</tr>
<tr>
<td>SDL</td>
<td>Specification and Description Language</td>
</tr>
<tr>
<td>SMEs</td>
<td>Small and medium enterprises</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL Protocol and RDF Query Language</td>
</tr>
<tr>
<td>SWRL</td>
<td>Semantic Web Rule Language</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>WCAG 1.0</td>
<td>Web Content Accessibility Guidelines 1.0</td>
</tr>
<tr>
<td>WCAG 2.0</td>
<td>Web Content Accessibility Guidelines 2.0</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>XHTML</td>
<td>Extensible Hypertext Markup Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Scope

The ultimate target of this deliverable is to present the initial architecture of the ACCESSIBLE system. The final version of the architecture will be delivered at month 18 through the deliverable D2.2.b [1]. The output of the work performed in all multidisciplinary tasks of the WP2 and WP3, such as the definition of technical specifications and system requirements, the state of the art in relevant technologies and the preliminary definition of application scenarios for the ACCESSIBLE system have been taken into account in order to result in the design of the overall architecture of the system.

Specifically, it combines the outcomes derived from:

- The User Needs, which are identified within T2.1 and specified in deliverable D2.2a “User needs and System Requirements Specification” (due for month 6),
- The State of the Art in Accessibility technologies and tools, that has been defined in deliverable D2.1 “State of the Art Survey in Accessibility Research and Market Survey” (due for month 6 and finally,
- The different ACCESSIBLE application scenarios, which are defined within WP2 and are specified in task T2.3 “Use cases and application scenarios”
- The system requirements defined in task 2.3 “Accessibility, assessment simulation and system requirements”
- The ACCESSIBLE application scenarios, which are defined within WP2 and are specified in task T2.4 “Use cases and application scenarios”

The overall architecture of a system consists of a set of architecture views that define the structure of the system, and how the subsystems relate and interact with each other. Thus the following architectural framework has been adopted:

- **Contextual Architecture and Scope** that describes the high-level design concepts, ideas, aims and motivations, together with their possible grouping of the functions into physical units.
- **Conceptual Architecture and Subsystem Functionality** that describes the high-level functions and sub-functions of the system, the flow of data between them, and the functionality of the ACCESSIBLE platform and its modules, but not in enough detail to produce a full logical scheme.
- **Logical Architecture** that describes the flow of data between the system modules both in terms of function calls between subsystems, message sets etc., based on sequence diagrams that follow the modelling of the definitive Use Cases that have been adopted by the project.

The report begins (section 2) with the methodology that has been adopted for the definition of the ACCESSIBLE architecture. In addition to this methodology, the impact of ACCESSIBLE requirements (as defined in D2.2.a, and D2.2.b) to the architecture is included to section 2. Section 3 presents the conceptual architecture of the ACCESSIBLE system as well as the logical architecture of ACCESSIBLE.
modules. This architectural view defined in section 3 sets the stage for the more detailed definition of ACCESSIBLE modules to follow within D3.3.b deliverable for month 18.
2 Methodology
This section describes the methodology followed to derive ACCESSIBLE system architecture.

2.1 Objective and Definition of System Architecture
System Architecture is the set of all the individual architectures, and any other statements, that describe the essence and constitution of the system. A System Architecture is a description which forms the basis for a class of systems and hence for a set of designs. The general characteristics of Architecture are the following:

- The Architecture must be open; this means that all suppliers, operators and users will be able to make use of what is in the Architecture.
- The architecture should be designed to apply to all different components that is composed of.
- The Architecture should be technology independent and promote (when it is possible) the use of generic solutions for which several technologies are available.

Conclusively, it should be noted that architecture is something at a higher level than a design, such that whilst it remains constant, many different designs can conform to it. Inside the ACCESSIBLE System there are (at least) three levels of architecture that are distinguishable between them. The aforementioned levels are shown in Figure 1.

![Figure 1 Model of the ACCESSIBLE System architecture](image)

The Level 0 architecture addresses the requirements of the system. These requirements are based on the analysis of user needs, application scenarios and the project’s concept. In this level, functional and technical requirements for the subsystems and tools of the ACCESSIBLE system are also defined. System requirements for the ACCESSIBLE tools and services are described in Deliverables D2.2.a “User needs and System Requirements Specification (first)” and D2.2.b “User needs and System Requirements Specification (final)” of ACCESSIBLE project.

In next Level 1, the overall system architecture is defined. An analysis for the system is described by presenting two available views of the system, namely the Static and the Dynamic model. The Static model of the system describes the objects (base classes, etc.) of the ACCESSIBLE system and their interfaces. Main purpose of this
model is to define the behavior of the modules that compose the system. On the contrary, the Dynamic model presents an analysis of the use cases of the system and the way that each module of the system acts within them. In this model, available use cases are also defined with the use of UML [2] diagrams (activity and sequence diagrams).

The Level 2 describes a set of architectures that define how the subsystems relate to each other.

For each ACCESSIBLE module of the system at least three separate individual architectures exist:

- Conceptual Architecture – this describes in a high level the components of each module of the system.
- Physical Architecture - this describes the grouping of the functions into physical units, reports the software and hardware requirements of each component of the system and the communication lines between them.
- Communication Architecture - this describes the flow of data between the physical units of the ACCESSIBLE sub-modules (developer-design aid modules, Web assessment module, Mobile assessment module, etc.).

Conclusively, the Level 2 can be considered as a detailed analysis and design of each component of the system. It is a manifestation of the Level 1 analysis with each sub-system and component being fully described.
2.2 Architecture development process

In this section, the overall methodology followed for the preparation of the current deliverable is presented. The procedure that pursued to derive the ACCESSIBLE system architecture is illustrated in Figure 2.

![Figure 2 Procedure to derive the ACCESSIBLE System architecture.](image)

As seen from Figure 2, the initial step prior to the analysis and design of the system architecture is to define and quantify the ACCESSIBLE system requirements (T2.3). System requirements are based on the analysis of User needs (T2.1), use cases and application scenarios, arising from T2.4, and T2.2 “Literature review and market survey”, respectively.

The use cases and scenarios present the system actors and the system behaviour. On the other hand, the system requirements specify the way the use cases and scenarios will be achieved. Finally, ACCESSIBLE system architecture is derived through the analysis and design of the system. The analysis and design of the system is reported in this deliverable while the scenarios of use and system requirements are analytically reported in the deliverables D2.2a [3], D2.2b, D2.3 [4] and D2.4 [5] of the ACCESSIBLE project.

Thus the ACCESSIBLE architecture is based on system requirements gathered from existing deliverables (D2.2a, D2.2b), where work is in progress. All requirements were identified and classified into specific categories (e.g. functional, non-functional, etc.). In some areas that are important for architectural design, there was not enough information. Therefore, architectural assumptions were set up and recorded in advance of architecture design (following section 2.6). The ACCESSIBLE architecture is based on the gathered requirements and assumptions and defined as component architecture with respect to initial the ACCESSIBLE infrastructure components described in DoW
2.3 Process for the development of the ACCESSIBLE System

2.3.1 General Description

Project risks include the size and complexity of the system and the use of new technologies, which are unfamiliar to most of the development teams. The provision and use of a development process to assist development and management teams can provide mitigation of this risk.

Aside: Deficiencies in the Traditional Lifecycle Model [6]
The waterfall model is generally understood to be a serial process comprising the following steps:

- Requirement capture (Analysis)
- Design
- Implementation
- Verification and Validation
- Deployment

This model has been effective in highly structured organizations where requirements change slowly, and where there is sufficient budget and resources allocated to the support functions. This scenario is usually found in highly regulated applications where risks of failure of the system are well-understood and significant.

Recent advances in software technology, specifically the advent of modelling tools and languages now make it possible to address the limitations of the waterfall model through a new technique called iterative development.

An iterative development process entails a series of short cycles through a “waterfall-like” process. Initial requirements are identified through standard methods of use case analysis, user interviews, examination of existing systems and so forth. A subset of these requirements are then expressed in a model or set of models built with one of the rapid prototyping tools now commercially available. The model(s) are then evaluated against the initial requirements. The initial model(s), analysis results and new or additional requirements are combined to serve as a baseline for another iteration of modelling. The expectation is that the next iteration of the model would cover a greater percentage of the requirements; hence more closely resemble the planned system. This process repeats until the behaviour of the system is expressed in sufficient detail in the model that implementation becomes a matter of software assembly instead of on-the-fly design.

As this process continues, the live experience with the model will cause some requirements to stabilize. Those stable requirements can then be “frozen” as baselines for development of the actual system, and actual development of the system can commence. Since the development is taking in smaller increments against clearly understood requirements and is based on a working model, the effort is better understood, easier to manage and has a greater likelihood of being completed on time with fewer defects and less rework.

The advantages of this approach are that requirements can evolve throughout the modeling process. The behavior of the system can be realized and understood more fully early enough in the process where resulting requirements changes can be integrated as a matter of course instead of as an exception after the fact.

Depending on the architecture of the system and requirements of the user community, it is possible to release the completed increments of the build process into production (internal or external) as standalone deliverables. This is referred to as Incremental Delivery.

Figure 3: Aside Deficiencies in Traditional Lifecycle Models

The advances in software tool technology has given rise to new development methodologies which address the limitations of the traditional “waterfall model”. The
The primary limitation of the waterfall model is the difficulty with timely capture and implementation of changes to requirements.

One of the most important aspects of the development process is the management of risk. Risk areas include for example, integration issues, unfamiliar technologies and resource availability. Risks should be actively sought out and addressed up front, rather than deferred.

In practical terms, a risk management approach involves identifying and solving high-risk issues in earlier iterations. In other words, early iterations of the project should focus more on eliminating unknowns and less on delivering a “laundry list” of features based on possibly faulty assumptions. With the majority of unknowns resolved early in the project, the later iterations will be more predictable and can proceed according to a well-defined schedule.

A significant class of risk comes from deferring architectural analysis until after the user requirements are fully defined and modelled. We believe that separation of technical architecture from user and application requirements into a ‘two track’ approach as described below assures that architectural risks are brought forward and managed as described above. In this approach, separate tracks are defined to allow architecture to represent a specific focus within the project independent from functional concerns for an application required by the business.

The four principles of the approach to process are summarised in the table 1 below.

<table>
<thead>
<tr>
<th>Principle</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental and Iterative Development.</td>
<td>This provides visibility for management, allows requirement changes along the way, and would potentially allow intermediate versions of the system to be delivered.</td>
</tr>
<tr>
<td>Risk-Oriented Planning.</td>
<td>Each increment should focus on the list of risks to be evaluated and managed, rather than on producing functionality while still leaving major unknowns. The deliverable of an increment is not just working code, but also the reduction of risks identified before planning of the increment.</td>
</tr>
<tr>
<td>Separate of technical and functional architectures.</td>
<td>Following the two track approach, separate tracks are defined to allow architecture to represent a specific focus within the project independent from functional concerns for an application required by the business. A common cause of failure in complex projects is the focus on business functionality, with architecture risks only managed after this functionality is fully defined and modelled. This defers these risks towards the end of the project. The separate architecture track should instead mitigate these risks as early as possible.</td>
</tr>
<tr>
<td>Continuous change management.</td>
<td>Since the requirements are generally continual evolving and subject to both internal and external project drivers requirements capture cannot be a one-shot activity performed at the beginning of the project. This requires a continuous requirement management approach coupled with a formal request for modification process. That is tailored to the constraints and methods that are introduced in component based development.</td>
</tr>
</tbody>
</table>

Table 1: Principles to approach development process
This approach proposes a development approach, which:

- Fills the semantic gap between user analysis, needs and software development.
- Encourages the reuse of core functionality across applications.
- Focuses on architecture to enable the flexible upgrade of pieces of the system.

### 2.3.2 Architecture Planning within ACCESSIBLE project

The component based development approach supports efficient by providing structure to assist risk-driven incremental planning avoiding high-risk approaches as described below.

The first high-risk planning approach is the “ultimate delivery” syndrome. That is every activity need to be fully completed before starting a new one. Generally this kind of strategy leads to no delivery. Furthermore it is probably not an achievable goal, because the every “finalise” activities provide some feedback that may have impact on activity itself. This can lead to an endless improvement circle. Another key point is that this strategy does not consider the changes that are likely to occur during the project life. In summary, this approach consumes a lot of time.

The following diagram illustrates this kind of planning.

![Figure 4: Waterfall “Ultimate Delivery” Syndrome](image)

The second one is the “massive parallelism” syndrome. That is, every activity is run in parallel in order to deliver the product sooner. Generally, this kind of strategy generates a lot of deadlocks because some activities need results from others. In all cases the delivery date is unpredictable. The picture below illustrates this kind of planning.

![Figure 5: Massive Parallelism” Syndrome](image)
In order to avoid these two syndromes, relevant actions will be taken through ACCESSIBLE software development processes.

<table>
<thead>
<tr>
<th>Syndrome</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Ultimate delivery”</td>
<td>Incremental and iterative process</td>
</tr>
<tr>
<td>“Massive parallelism”</td>
<td>Check and coordinate the dependencies between tasks and work packages</td>
</tr>
</tbody>
</table>

Table 2: Lifecycle Risks and Actions

These considerations drive towards the iterative and incremental planning model pictured below (figure 6) that will be followed by ACCESSIBLE software and development teams. It is important to notice that the different activities can be conduct by different teams with corresponding skills.

![Diagram](image)

Figure 6: Incremental and iterative process within ACCESSIBLE

2.4 Architectural description in conformance with IEEE standards......

The architectural description of ACCESSIBLE is in conformance with family of standards IEEE 1471 [7]. IEEE 1471 is the short name for a standard formally known as ANSI/IEEE 1471-2000, Recommended Practice for Architecture Description of Software-Intensive Systems. In 2007 this standard was adopted by ISO/IEC JTC1/SC7 as ISO/IEC 42010:2007 [8], Systems and Software Engineering -- Recommended practice for architectural description of software-intensive systems. [1]

The IEEE 1471’s contributions can be summarised as follows:

- It provides definitions and a meta-model for the description of architecture
- It states that an architecture should address a system's stakeholders concerns
- It asserts that architecture descriptions are inherently multi-view, no single view adequately captures all stakeholder concerns
- It separates the notion of view from viewpoint, where a viewpoint identifies the set of concerns and the representations/modeling techniques, etc. used to describe the architecture to address those concerns and a view is the result of applying a viewpoint to a particular system.
- It establishes content requirements for architecture descriptions and the idea that a conforming architecture description has a 1-to-1 correspondence between its viewpoints and its views.

IEEE 1471 provides some universal recommended practice and a conceptual model of architectural description, depicted in the following figure 7.

Figure 7: Illustration of the conceptual framework of IEEE 1471 standard for Architecture Description

The most important concepts of the IEEE 1471 standard meta-framework are system, stakeholder, architectural description, view, and model. The focus of IEEE1471 standard is software intensive systems - Systems in which software interacts with other software, systems, devices, sensors and with people- includes computer-based systems ranging from individual software applications, information systems, embedded systems, etc. Therefore, it clearly applies to ACCESSIBLE.

IEEE 1471 defines a set of normative requirements for conforming architecture descriptions, including the following:

- Architectural Description identification, version, and overview information (clause 5.1 of the recommended practice)
• Identification of the system stakeholders and their concerns judged to be relevant to the architecture (clause 5.2)
• Specifications of each viewpoint that has been selected to organize the representation of the architecture and the rationale for those selections (clause 5.3)
• One or more architectural views (clause 5.4)
• A record of all known inconsistencies among the architectural description’s required constituents (clause 5.5)
• A rationale for selection of the architecture (clause 5.6)

The ACCESSIBLE architecture will be evaluated upon its completion against these requirements to be able to judge on its conformance with IEEE 1471.

2.5 System Concept

The development of accessible software requires specialized knowledge and significant effort from developers and designers. With the additional encumbrance of taking into account different kinds of accessibility requirements, guidelines and best practices, and different implementation technologies (which by themselves might pose severe problems of delivering accessible applications), developers are faced with a daunting task. Therefore, the highly specialised skills required for developing accessible software sets aside most developers. To mitigate these challenges, developers can be guided during the development process in the creation of accessible ICT. To do this, developers need to have a conceptual framework in which to situate disabled-related guidelines, which they often do not have due to lack of experience with the needs of people with disabilities and the access technologies they use.

The ACCESSIBLE project will try to implement specific methodologies and tools for ensuring accessibility for designers and software developers (figure 8). Thus it envisages the development of a wide variety of open source tools for the accessibility assessment of different application domains such as Web, mobile, Web services and description languages, supported through a plug-in fashion (e.g., Web accessibility would be one of the plug-ins), based on semantic technologies, ontologies and SWRL (Semantic Web Rule Language) rules.

Since impairments are different and methodologies and techniques for solving the accessibility problem for the particular target user groups are complex and various, even experienced UI developers require guidance and assistance. Ideally, this help should be supported by the integrated development interface (IDE) they are utilizing for the implementation or through easy-to-use open source tools that can be used during the whole lifecycle of their software developments.

Different architectural approaches are relevant for the ACCESSIBLE architecture as identified below:

• Open source standalone applications that can be downloaded to users terminals and can be used for the accessibility assessment of their software components
• Web Services that can be offered through the ACCESSIBLE portal for users that they would prefer to verify their applications online
• Plugin based developments that should be supported by common integrated development Environments (e.g. Netbeans [9]) that developers and designers are utilizing for their software developments. These plugin components can be used for integrated guidance and assistance for the accessibility design and development of software applications.

![Figure 8: ACCESSIBLE conceptual functionalities](image)

## 2.6 Impact of Requirements to the ACCESSIBLE Architecture

This section takes the most important ACCESSIBLE requirements specified in D2.2.b in order to analyze how well the architecture already supports the identified requirements, and where the architecture needs to be adapted to make it possible to deliver the required functionalities. It is to be noted that a single requirement may address multiple architectural components and a single architectural component typically has numerous non-trivial relations to various requirements. This section will allow us to ensure that the architecture is not in conflict with any of the ACCESSIBLE requirements, which is important for ACCESSIBLE in order to enable the architecture to correspond to the holistic view of the project.

The following tables are structured in the following way:

- **Heading**: The original Requirements name as described in D2.2.b
- **Requirement**: A brief description of the requirement taken from D2.2.b for a better readability
- **Architecture Analysis**: The analysis of this requirement in relationship to the ACCESSIBLE architecture. This describes the impact of the requirement to the overall architecture and shows how ACCESSIBLE will address the requirement by its components.
2.6.1 ACCESSIBLE Generic Requirements

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ1-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The system shall support developers and designers to assess the accessibility of their developed applications via the ACCESSIBLE assessment module.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>This requirement is covered within the ACCESSIBLE architecture by the Assessment Simulation modules that use the required information from the ACCESSIBLE ontological framework. Thus, the architectural approaches that will be supported for the Assessment simulation modules will be Open source standalone applications that can be downloaded to users terminals and can be used for the accessibility assessment of their software components, Web Services that can be offered through the ACCESSIBLE portal for users that would prefer to verify their applications online and Plugin based developments that should be supported by common integrated development Environments such as Netbeans.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ1-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The system shall provide users with detailed accessibility instructions and guidelines for the implementation of accessible Web, Mobile applications, Web services and/or description languages.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The ACCESSIBLE architecture contains detailed accessibility guidelines and instructions based on different standards for the implementation of accessible Web applications, mobile applications, Web services and description languages. The aforementioned information can be accessed by the ACCESSIBLE OWL ontologies through the ACCESSIBLE rules editor and through the ACCESSIBLE portal.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ1-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Where appropriate, the system shall allow the definition of accessibility terms and constraints using ontology based description language and different domain ontologies.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>Accessible Ontologies aim at providing support for the formal and unambiguous definition of accessibility domains, as well as the possible semantic interactions between them. Generic and domain OWL Ontologies will describe the division between generic and domain-specific accessibility concepts, and how they can be mapped within accessibility assessment scenarios. The ontologies formalise conceptual information about: (1) The characteristics of users with disabilities, devices, applications, and other aspects that should be taken into account when describing an audience with disabilities and developing software applications; (2) Accessibility standards and encompassing assessment rules and checkpoints; and (3) Mapping requirements and constraints of users to assessment rules.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ1-4</th>
</tr>
</thead>
</table>
## Requirement

**The system shall enable an easy reconfiguration of accessibility assessment processes**

### Architecture / Analysis

The ACCESSIBLE architecture in general and more specifically the assessment module is wrapped around the ACCESSIBLE knowledge layer (ACEESIBLE ontologies) a Semantic Registry and Repository for storing and retrieval accessibility terms and assessment constraints based on SWRL [10] rules. The SWRL editor will allow the retrieval of information instances by providing a functional and clear interface based on open standards and technologies such as SWRL and OWL.

## Heading

### G-REQ1-5

**Requirement**

The system shall provide appropriate tools to developers and designers to simulate their developed applications with visual impairments via the developer-designer aid module.

### Architecture / Analysis

The ACCESSIBLE architecture contains an own component – the Developer-design aid module Editor - to perform this task. The purpose of this module is to assist developers and designers to better empathise with those who have reduced capabilities, and to help understand how capability loss affects the ability to interact with software applications and services. Self-learning software applications, which can be obtained as a NetBean IDE plugins and/or as standalone applications, will be part of this module.

## Heading

### G-REQ1-6 - G-REQ1-8

**Requirement**

The system shall take into account all the most widely known Web accessibility standards, such as WCAG 1.0 [11], WCAG 2.0 [12], Section 508 [13], etc and also modify the existing ones or add new ones - The system shall take into account some of most widely known mobile accessibility standards, such as MWBP 1.0 [14], etc. and also modify the existing ones or add new ones

### Architecture / Analysis

One of the main issues in designing and developing the proposed ACCESSIBLE assessment framework was to make it maintainable and extensible, while assuring model consistency within the framework. Therefore, we have separated the Accessible Ontology into three distinct dimensions: Generic Ontology, Domain-specific ontologies and Rules Ontology. The expressive power of the OWL language through standards domain ontologies (different OWL ontologies for WCAG 1.0, WCAG 2.0, Section 508, MWBP) will be sufficient to derive good search results, mappings and later transformations or additions. SPARQL [15] language is being used for processing the correspondences.

## Heading

### G-REQ1-7

**Requirement**

The system shall support any potential user to select one or more of the stored accessibility standards or guidelines, such as WCAG 1.0, WCAG 2.0, and Section 508.
Architectural Analysis

The ACCESSIBLE portal provides to users appropriate visualisation functionalities in order to access and view the stored accessibility standards within the ACCESSIBLE ontologies. In addition to this, the standalone version can support the visualisation of preferable standards. The Rules Inference Engine will be responsible for generating the inferred knowledge from the ontology and make it available to the other modules of ACCESSIBLE.

### G-REQ1-9

**Requirement**

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

**Architectural Analysis**

The Rules Inference Engine will provide runtime support to all ontologies and accessibility evaluation of different technologies, as created in the context of ACCESSIBLE. This Engine affords the binding between ontologies and software components that perform accessibility evaluation, through the use of meta-programming techniques available in the Java programming language. The Rules Inference Engine leverages the power of Semantic Web languages (RDF [16], RDF-S [17], OWL [18], and SWRL) and related open-source technologies (Jena Semantic Web Framework and Pellet OWL Reasoner) in order to provide support to users for the selection of preferable accessibility criteria and browsing all content in a user-friendly way.

### G-REQ1-10

**Requirement**

The ACCESSIBLE system shall support the implementation of appropriate rules based on the ACCESSIBLE knowledge resource and on user’s preferences.

**Architectural Analysis**

The rules editor component, allows the user to manage SWRL rules between different elements within ACCESSIBLE ontologies.

### G-REQ1-11 - G-REQ1-12

**Requirement**

The system shall support developing/assessing for a variety of disabilities and functional limitations (based on the ICF categorization). At least visual, auditory, and hearing disabilities should be taken into account.

**Architectural Analysis**

The expressive power of the OWL language through functional general characteristics and disabilities domain ontologies (different OWL ontologies for impairments, disabilities, ICF [19] functional limitations and capabilities) will be sufficient to derive good assessment functionalities that cope with user’s disabilities. The user interface of the stand-alone tools as well as the ACCESSIBLE portal will allow the user to make specific selections concerning the evaluation process using the inferred knowledge that will have been produced by the Rules Inference Engine.
<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ1-13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Where appropriate, each user preferences shall combine different kind of disabilities, assistive technologies and accessibility standards.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>Since OWL is used for ACCESSIBLE knowledge representation concerning standards, disabilities, and assistive technologies, an addition of combination is easily possible. The user will be able to view and manage those combinations in the ACCESSIBLE rules editor component through SWRL language. Each user can manually change or annotate existing entries or to create new rules.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ1-14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Where appropriate, the developers or the system users shall receive appropriate reports concerning the results of their accessibility tests based on the EARL language [20].</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>This requirement is covered by the ACCESSIBLE EARL-based reporting tool providing users with appropriate accessibility assessment results. In addition to this relevant accessibility metrics like WAB will be incorporated to the reporting tool.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>P-REQ1-1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Average system response time shall be in proportion with the complexity of the objects that are tested.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The response time depends mainly on complexity of application objects (e.g. how many web pages will be tested) and the reaction of the ACCESSIBLE knowledge storage. As this storage is expected to be designed through usage of OWL ontologies and SWRL rules, assembling of different constraints and characteristics, this requirement must be managed at the implementation of SPARQL queries. Within the ACCESSIBLE implementation, it needs to be ensures that the number of queries that need to be implemented is minimized and that all requests are optimized in order to minimize the response time.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>P-REQ1-2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>ACCESSIBLE assessment modules shall support hundreds of simple queries per minute, where simple queries are defined as access to centrally ontology-stored data using typical, predefined rules.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The Rules Inference Engine leverages the power of Semantic Web languages (RDF, RDF-S, OWL, and SWRL) and related open-source technologies (Jena Semantic Web Framework and Pellet OWL Reasoner) in order to provide support to users for the selection of preferable accessibility criteria and browsing all content in a user friendly way. SPARQL language is being used for processing the correspondences which can support hundreds of simple queries per minute.</td>
</tr>
</tbody>
</table>
**Heading** | **O-REQ1-3, R-REQ1-1 - O-REQ1-1**  
---|---  
**Requirement** | Most of ACCESSIBLE modules shall provide diagnostics messages in case of unsuccessful or uncertain operations.  
**Architecture / Analysis** | Within ACCESSIBLE architecture adequate measures will be taken to ensure the overall robustness of different components, both under normal operation and under exceptional circumstances. Thus a help menu will be incorporated to the system, that will facilitate the user understand the operation of the system, and guide him along the different assessment processes. The ACCESSIBLE system will adopt:  
  - Error recovery procedures: Error prevention will be enacted through the application of standard procedures at different levels.  
  - Error messages: Exceptional behaviour of the system has to be transparent for the user at two levels:  
    - Application (end-user) level. End-user to be notified about the result of the performed operation. Especially it is important concerning abnormal performance. The content of the provided message should be simple, clear, exact and understandable for the user.  
    - Administration level. The first step is to notify the corresponding administrator of the ACCESSIBLE portal and appropriate services. (in standalone version end user is also the administrator). The next step is to provide the administrator with the appropriate means to examine the problem, to perform appropriate measures for error recovery and (eventually) to make the end-user aware about the reason for the error.  

**Heading** | **O-REQ1-4**  
---|---  
**Requirement** | All system components shall be easy to navigate to people with different knowledge and capabilities (developers, designers, testers, etc.)  
**Architecture / Analysis** | The expenses, influencing ACCESSIBLE access could be divides into several types. Following, the role of the architecture in decreasing expenses, is described:  
  - ACCESSIBLE Web services maintenance expenses. Services have to be arranged in a way that they allow administrators to configure and maintain the different components in a consistent and comfortable way.  
  - Efforts have to be spent for developing convenient, simple, self-installed and self-configured ACCESSIBLE components, in order to avoid difficulties, caused by the lack of user’s technical skills.  
  - ACCESSIBLE consultancy. When designing ACCESSIBLE components, attention has to be paid on ensuring effective means for discovering and solving problems, especially on those, causing from incorrect user’s work.
<table>
<thead>
<tr>
<th>Heading</th>
<th>M-REQ1-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The ontology schema will provide some extensibility to ACCESSIBLE components in order to address both the heterogeneity of the data and the fact that new forms of data may appear after the duration of the project.</td>
</tr>
</tbody>
</table>
| Architecture / Analysis | The system can easily acquire and adapt features supporting the augmentation of existing functions and the deployment of new ones alongside the existing. The ability to extend the ACCESSIBLE ontologies (by the developers, but possibly by the users too) beyond the initial design and through the embedding of modules possibly developed by third parties or through a public API scheme (like the ontology editor Protégé) can be very important to allow participation by the general public in the system development effort by tailoring the system for their needs.  
- Third parties and end users will be able to extend the functionality by introducing new rules through the ACCESSIBLE rules editor.  
- The ACCESSIBLE OWL ontologies allow managing data dependent functionality by maintaining a large variety of data and metadata. |

<table>
<thead>
<tr>
<th>Heading</th>
<th>M-REQ1-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>All software assessment modules developed in the project will be offered through Web services from the ACCESSIBLE portal.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>Some the assessment simulation modules (like the web applications assessment module) will be developed as standalone applications but additionally all the assessment simulation modules (web, mobile, web services) will be offered through web services and will communicate with the Web Interface of the Accessible Portal.</td>
</tr>
</tbody>
</table>

### 2.6.2 ACCESSIBLE Assessment Simulation Module Requirements

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The assessment simulation module shall test the accessibility of applications according to the guidelines of widely accepted accessibility standards (e.g., WCAG1, WCAG2, Section 508, etc) where appropriate.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The original structure and instances of widely accepted accessibility Web standards are imported by the usage of OWL format through the Protégé tool. Thus, different domain ontologies per each accessibility standard can be imported to the ACCESSIBLE generic ontology. To elaborate an evaluation framework for a priori testing, and based on the harmonized methodology of WP3, relevant methods and a number of criteria, parameters and SWRL rules, will be incorporated to the ACCESSIBLE ontologies with the usage of protégé and appropriate plugins (e.g. SWRL tab). These rules will support the properties and relationship’s definitions between generic and domain ontologies. In case that the original data structure or instances are changing, the administrator user will need to use Protégé to modify the relevant data. The introduction of relevant generic terms (based on WCAG format) within the ACCESSIBLE generic ontology can enhance the import of new domain ontologies (e.g modification of a stored standard and/or the introduction of a new one). The Rules Inference Engine will pass all the inferred knowledge of the ACCESSIBLE ontology to the Web Interface of the ACCESSIBLE Portal (or the GUI of the standalone version) and the user will be able to make the desired selections (ex. standards, impairments, disabilities, functional limitations, etc) regarding the evaluation process.</td>
</tr>
</tbody>
</table>

| Heading | G-REQ2-2 |
| Requirement | All assessment simulation modules shall connect with the ontology and run all the SWRL rules defined in the ontologies. |
| Architecture / Analysis | All the assessment simulation modules will communicate with the Rules Inference Engine, which be able to run all the SWRL rules defined in the ontology or execute SPARQL queries in order to retrieve specific knowledge and finally the inferred knowledge will be passed to the appropriate components of the assessment simulation modules. |

<p>| Heading | G-REQ2-3 |
| Requirement | All assessment simulation modules shall be able to get the instances of all the classes defined in the ontologies and present them in an appropriate GUI. |
| Architecture / Analysis | All the assessment simulation modules will communicate with the Rules Inference Engine, which be able to execute SPARQL queries in order to get the instances of any class defined in the ontology. The SPARQL queries may be hard-coded in the source code of each assessment simulation module or in separate .sparql files. Finally, the result of the SPARQL queries (the standards/impairments/disabilities/functional limitations, etc. being described in the ontology) will be presented in the graphical user interface of each assessment simulation modules (or in the Web Interface of the Accessible Portal). |</p>
<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The assessment simulation module shall give user the opportunity to define the tests that will be executed according to specific selections made in some pre-defined categories (user, device, disability, functional limitation, application, etc) as these categories and their relationships defined in the ontology.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The Rules Inference Engine poses no obstacles to the selection of tests, thus allowing the tailored execution of accessibility assessments, as defined in the ACCESSIBLE ontology. The Rules Inference Engine will generate all the inferenced knowledge of the ontology including all the instances of the classes as well as their correlations. A part of the inferenced knowledge (the instances of the most important classes ex: standards, disabilities, etc) is going to be chosen to be presented to the user and the user will have the opportunity to make his selections that will affect the approaches which will be executed during the evaluation process.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The web application assessment tool shall support the accessibility assessment of HTML Web applications.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The W3C Markup Validator will be responsible for returning the errors/warnings concerning the structure found in a HTML document and the Web Accessibility Evaluator using the HTML Parser will execute all the selected by the user approaches in order to perform the accessibility assessment process.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The web application assessment tool shall support the accessibility assessment of (HTML) Web applications generated from different server-side environments, languages or technologies (PHP, ASP, JSP, etc.)</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>In case that PHP, ASP or JSP technologies are being used for the development of a web site/application, the PHP, ASP, JSP code is being executed on the server and finally the server returns the corresponding HTML code to the client (in our case the web applications assessment tool). Thus, it is obvious that as the web applications assessment tool will support the evaluation of HTML applications it will be able to support PHP, ASP and JSP applications too.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>In the ACCESSIBLE Web assessment module, the parsing of the XML-based documents (XHTML, etc) will be performed using Groovy scripts.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>In the Web Applications Assesment Module, the HTML Parser will use Groovy scripts in order to perform the parsing of the HTML/XHTML documents.</td>
</tr>
</tbody>
</table>
### Heading G-REQ2-8

**Requirement**

In the ACCESSIBLE assessment modules, the SPARQL queries that will be executed will be defined in separate text files with .sparql extension.

**Architecture / Analysis**

The Rules Inference Engine that will communicate with every assessment module will be able to execute SPARQL queries that are hard-coded in the source code of each module or in separate text files with .sparql extension. For a more flexible approach of the modules’ development, the use of separate .sparql files is more preferable.

### Heading G-REQ2-10

**Requirement**

The mobile assessment module shall support the accessibility assessment of the mobile Web.

**Architecture / Analysis**

The architecture of the mobile accessibility assessment tool takes into account the particularities of mobile Web applications.

### Heading G-REQ2-11

**Requirement**

The mobile assessment module shall verify how interaction input is performed in the application, and how it influences accessibility issues.

**Architecture / Analysis**

The architecture of the mobile accessibility assessment tool does not pose any constraints on interaction input analysis.

### Heading G-REQ2-12

**Requirement**

The Web Services assessment tool should be able to evaluate accessibility features of the most common types of Web Services worldwide.

**Architecture / Analysis**

To the present, the most commonly used types of Web Services are the ones following the W3C SOAP protocol standardization. Moreover, these usually follow the “rpc/encoded” and “document/literal” formats, defined within the W3C SOAP protocol specification [21]. By incorporating a Web Services Parser based on the Apache Axis1 and Axis2 SOAP implementations, the Accessible Web Services Assessment tool has the ability to parse both these two; most commonly used types of Web Services. Information acquired from this parsing process is thereafter used from the tool for the evaluation of WS Accessibility Guidelines, as these are defined within the Accessible Ontology.

### Heading G-REQ2-13

**Requirement**

The Web Services assessment tool should enable the integration of the modified ASK -IT alignment tool.
The Web Services assessment tool utilizes the capabilities offered from the ASK-IT alignment tool in order to acquire more information regarding the services under evaluation. For the purposes of the ASK-IT project, services were aligned through the ASK-IT alignment tool to Conceptual Service models defined within the ASK-IT Ontology. For the purposes of the Accessible project, these service models have been replaced with the set of the Accessible “Ideal Operations” defined within the Accessible Ontology.

After assessing a first set of Web Services Accessibility Guidelines automatically, the tool forwards the user (evaluator) to the web-based ASK-IT Service Alignment tool. At this point, the user is asked to align the operations defined within the service under evaluation to a set of “Ideal Operations” defined within the Accessible Ontology. The information retrieved from this alignment process is stored in a MySQL database. The Web Services assessment tool contacts this database and acquires all information derived from the alignment process. This information is finally used from the tool for the evaluation of a further set of Web Service Accessibility Guidelines.

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The Web assessment tool should enable the integration of the SET tool for the visualisation of the assessment results.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The GUI of the standalone version of the web applications assessment tool will cooperate with the SET tool in order to achieve a better visualization of the results of the evaluation process that the Web Accessibility Evaluator will return.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The Web Services assessment module will be able to evaluate accessibility features of Web Services, using the SOAP-based WSDL files [22] defining them.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>Services that follow the W3C SOAP-based standardization are defined on the basis of the Web Services Definition Language (WSDL). Thus, each Service has a specific WSDL file which contains information regarding the structure and functionality of the WS (Operations, Inputs, Outputs, etc.). Some accessibility features of Web Services can be assessed by using information derived from the parsing of WSDL files. By incorporating a Web Services WSDL Parser based on the Apache Axis1 and Axis2 SOAP implementations, the Accessible Web Services Assessment tool has the ability to parse WSDL files defining Web Services and acquire all information that can be derived from this procedure. Consequently, by using this information, the tool is able to evaluate accessibility features of Web Services.</td>
</tr>
<tr>
<td>Heading</td>
<td>G-REQ2-18</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>Requirement</td>
<td>The Web Services assessment module should be able to read the types of WSDL files that are used by the majority of working SOAP-based Web Services.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>Until today, the most commonly used types of WSDL files defining Web Services are the rpc/encoded and document/literal ones. By incorporating a Web Services WSDL Parser based on the Apache Axis1 and Axis2 SOAP implementations, the Accessible Web Services Assessment tool has the ability to parse both rpc/encoded and document/literal WSDL files defining Web Services. In particular, the parsing of rpc/encoded WSDL files is based on the Axis1 framework, whereas the parsing of document/literal ones is based on the Axis2.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The user shall be able to select manually one by one all the tests that will be executed.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The Web Interface of the Accessible Portal and the GUI of the standalone versions of the assessment simulation modules will provide forms where the user will be able to select manually (ex. using checkboxes) any approach (primitive accessibility test) that will be executed during the evaluation process.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The assessment simulation module after the assessment completion shall output the results in EARL format</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>Every assessment simulation module will sent in XML format all the results of the evaluation process to the ACCESSIBLE Earl-based Reporting tool, which is responsible for the EARL-based accessibility report generation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The description languages assessment tool should be able to evaluate accessibility features for designs/systems based on the SDL-2000 standard</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The Description Languages Accessibility Assessment Module is able getting the assessment guidelines and parsing the SDL-2000 standard [23] source code files based on the rules corresponding to these guidelines. Finally it compares the pre-defined tresholds of the features with the application features to accomplish the accessibility assessment.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The description languages assessment module should be able to read and parse the types of SDL files that are based on SDL-2000 using an SDL parser.</td>
</tr>
</tbody>
</table>
Architecture / Analysis

The Graphical Editor of the SAFIRE tool which is used for the creation of SDL Systems that can be assessed by the Accessibility Assessment module also produces the the SDL System in text format file that contains the application source code according to SDL-2000 standards. The application source code can be parsed by the Description Languages Accessibility Assessment module.

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The assessment module shall work in two different modes for HTML applications: - single web page assessment - entire web site assessment</td>
</tr>
</tbody>
</table>

Architecture / Analysis

The user will be able to evaluate a single web page or an entire web site using the corresponding options of the GUI (for the standalone version) or the Web Interface of the ACCESSIBLE Web Portal (for the web service version). In case of an entire web site, the Web Accessibility Evaluator will cooperate with the Web Crawler in order to get the URLs of all the pages containing in the web site.

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>If the entire web application mode is being chosen, there shall be an option defining the maximum number of web pages that will be examined.</td>
</tr>
</tbody>
</table>

Architecture / Analysis

The Web Interface of the ACCESSIBLE Portal as well as the GUI of the standalone version of the Web Applications Assessment tool will give user the opportunity to define the maximum number of the pages that will be examined. Then, the Web Crawler will be informed in order to return the URLs for the specific number of pages.

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-26</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The assessment module shall work in two different modes for Mobile applications: - Mobile Web page automated assessment - Mobile Web site automated assessment</td>
</tr>
</tbody>
</table>

Architecture / Analysis

The architecture for the mobile accessibility assessment tool is able to cope with both page and site evaluation procedures.

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-27</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The description languages assessment module shall accept as input files with relevant file extensions (e.g. “<em>.pr”, “</em>.fsm”) for verifying the accessibility of SDL systems/designs.</td>
</tr>
</tbody>
</table>
The SAFIRE tool produces the application’s source code and saves it as a “.pr” or an “.fsm” file. These files are the inputs of the Description Languages Accessibility Assessment module in order to make the assessment and verify the system’s accessibility.

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ2-28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The description languages assessment module shall include a software sub-system capable to parse SDL designs based on the SDL-2000 standard.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The application source code which is based on the SDL-2000 standard can be parsed by the Accessibility Features Parser component of the Description Languages Accessibility Assessment module.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>P-REQ2-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The description languages assessment module shall be able to parse files containing complex SDL designs (consisting of many blocks, processes, states, inputs, outputs, timers etc.) at a reasonable time for the user and less than 3 minutes time.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>This requirement will be verified during the validation phase of the project.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>P-REQ2-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The description languages assessment module shall be able to parse files containing complex SDL designs (consisting of many blocks, processes, states, inputs, outputs, timers etc.) at a reasonable time for the user and less than 3 minutes time.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>This requirement will be verified during the validation phase of the project.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>P-REQ2-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>If a Web based version will be developed for the Web assessment, it is planned to support hundreds of simultaneous users.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The ACCESSIBLE Portal will be able to support hundreds of simultaneous users that will be able to use the Web Applications Assessment Web Service via the Web Interface of the portal.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>R-REQ2-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Where appropriate the assessment tools will be robust and shall recover from connectivity failures with other modules.</td>
</tr>
</tbody>
</table>
Architecture / Analysis | There may be cases that some of the components of the Web Applications Assessment Module will fail (ex. the W3C CSS Validator [24], which is an open-source tool used as a black-box by the Web Applications Assessment Module, in some cases may throw exceptions). For all the known possible problems, an error recovery mechanism will be developed, in order to make the whole evaluation process robust.

### M-REQ2-4

<table>
<thead>
<tr>
<th>Requirement</th>
<th>The description languages assessment module shall accept as input files (e.g. “<em>.pr”, “</em>.fsm”) that include SDL systems/designs based on SDL-2000 standard.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture / Analysis</td>
<td>The Graphical Editor of the SAFIRE tool which is used for the creation of SDL Systems that can be assessed by the Accessibility Assessment module also produces the the SDL System in text format file that contains the application source code according to SDL-2000 standards. The source code files are saved with extension “.pr” and “.fsm” and the Accessibility Assessment module takes them as input for the assessment.</td>
</tr>
</tbody>
</table>

### 2.6.3 ACCESSIBLE ontological Knowledge Resource Requirements

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ3-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The ontology shall support the SWRL (a Semantic Web Rule Language) rules, which form of an implication between an antecedent (body) and consequent (head).</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>SWRL rules are seamlessly integrated into OWL technologies, as SWRL itself is an OWL-based language.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ3-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>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.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>SPARQL queries are inherently tied to OWL technologies, thus allowing the fulfilment of this requirement.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ3-14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>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.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The architecture for the Rules Inference Engine reflects this requirement, by selecting the Jena and Pellet libraries for OWL querying (through SPAQRL) and rules inference (through SWRL).</td>
</tr>
<tr>
<td>Heading</td>
<td>P-REQ3-1</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>Requirement</td>
<td>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.</td>
</tr>
</tbody>
</table>

| Architecture / Analysis | The components selected for ontology parsing and manipulation (Jena and Pellet) within the Rules Inference Engine, are state-of-the-art OWL libraries, widely used and acclaimed for OWL-based software development. Consequently, response times of the Rules Inference Engine and of the access to the ACCESSIBLE knowledge resource are directly related to the limitation of the selected technology and Semantic Web features. |

<table>
<thead>
<tr>
<th>Heading</th>
<th>P-REQ3-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>In the same way, the inference engine in the complex rules and queries execution shall also take into account the performance.</td>
</tr>
</tbody>
</table>

| Architecture / Analysis | The selected components for SWRL parsing and inferencing (Pellet libraries) are linearly correlated with the complexity of the rules available on the ACCESSIBLE knowledge resource. |

<table>
<thead>
<tr>
<th>Heading</th>
<th>M-REQ3-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Developers shall be able to specify new SWRL rules and/or SPARQL queries through a special purpose GUI.</td>
</tr>
</tbody>
</table>

| Architecture / Analysis | The architecture of the Rules Inference Engine is supported by a GUI for the addition of new concepts to the ACCESSIBLE knowledge resource. |

<table>
<thead>
<tr>
<th>Heading</th>
<th>M-REQ3-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>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).</td>
</tr>
</tbody>
</table>

| Architecture / Analysis | All architectural components selected for the Rules Inference Engine are based on the Java language and are freely available as mature, open source products. |

<table>
<thead>
<tr>
<th>Heading</th>
<th>M-REQ3-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>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.</td>
</tr>
</tbody>
</table>

| Architecture / Analysis | The architecture of the Rules Inference Engine is totally decoupled from technology-centric accessibility assessment procedures and, therefore, poses no limits to its integration in the context of external accessibility assessment products and services. |
### 2.6.4 User Interface Portal Requirements

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ4.1-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The User Interface of ACCESSIBLE components should be user friendly that means that the users must locate the required information on the screen easily.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The user interface design iterations conducted ensure that the final user interface of accessible has been developed taking into account issues such as accessibility, user friendliness, help provision, intuitiveness etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ4.1-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The Web based Interface shall provide to users the ability to test web sites, web applications via their URL.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The user interface portal provides users with the option to structure their evaluation sessions in a number of projects. Each of these projects contains information about the application to be evaluated such as url, name, description etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ4.1-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Users shall have the option to download the standalone tools developed in the context of ACCESSIBLE (Net Beans IDE plug-ins, the accessible assessment and simulation module etc).</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The downloads section of the user interface portal has been defined as the on stop shop for getting access to the stand alone tools developed in the context of ACCESSIBLE and can be accessed through the main page offered to registered users.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ4.1-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The User Interface shall provide information about the goals of ACCESSIBLE project.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>In order for the visitors of the portal to have the option to get informed about the scope, objectives and goals of ACCESSIBLE a specific area has been defined that aims at providing such information. This section together with news, links and newsletters are incorporated for strictly informative purposes and are not related with the main evaluation functionality that is the main target of the portal.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ4.1-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The User Interface shall provide clear instruction before and during interaction.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>Via the usage of EAGER both the initiation of a function and its completion through the selection of an action are accompanied with help when a profile for a novice user is selected.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ4.1-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The User Interface shall provide high efficiency, reliability, and resource utilisation of the ACCESSIBLE tools.</td>
</tr>
</tbody>
</table>
The architecture of ACCESSIBLE has been structured to employ the state of the art on communication tautologies via the web using web services for propagating data back and forth from the user interface to its core components. Web services are efficient and reliable and can be proven fast enough and fault tolerant when deployed on a web farm with load balancing.

**G-REQ4.1-7**

**Requirement**
The User Interface shall provide all the useful information to web developers (such as standards and guidelines) in order to create accessible applications.

**Architecture / Analysis**
The user interface portal has been designed to incorporate both a digital library of resources on accessibility and an online training tool offering access to educational material regarding the usage of standards and guidelines in general but also specifically targeted to the tools designed and developed in the context of ACCESSIBLE.

**G-REQ4.1-8**

**Requirement**
Users must have the option to access related links of various accessibility issues and have the option to get informed about upcoming events relevant to the ACCESSIBLE.

**Architecture / Analysis**
This requirement is met via the integration of a links and events module. These modules can be accessed for the front page of the non-subscribed user area.

**G-REQ4.1-9**

**Requirement**
Where appropriate the User Interface shall be designed in accordance to the project’s logo and colours and contrast shall comply with the usability guidelines.

**Architecture / Analysis**
The user interface design conducted for the user interface portal meets the requirements for being in accordance with the project’s logo and the contract scheme used has been checked for compliance with the usability guidelines using colour contrast checking tools.

**G-REQ4.1-10**

**Requirement**
Users should have the option to subscribe to the portal providing their personal information and therefore get access to the facilities offered to subscribe users.

**Architecture / Analysis**
Access to the ACCESSIBLE facilities has been restricted during the design of the user interface portal only to subscribed users. To this end a registration mechanism has been developed prompting the user to register prior to using the ACCESSIBLE evaluation framework. This mechanism also facilitates our goal of storing user based information regarding evaluation sessions, project settings etc.

**G-REQ4.1-11**

**Requirement**
Registered users should have the option to access resources and training material related to the development and evaluation of accessible applications in various contexts.
This is achieved through the usage of the incorporated digital library providing access to resources such as collections, guidelines, best practice examples etc.

**G-REQ4.1-12**  
**Requirement**  
Users should have the option to perform keyword based search for resources.

**Architecture / Analysis**  
One of the main facilities incorporated on the digital library for offering fast access to the available resources is the keyword based search facility.

**G-REQ4.1-13**  
**Requirement**  
Registered users should have the option to personalise the way that knowledge is displayed through the existence of facilities such as “knowledge profiles”.

**Architecture / Analysis**  
The knowledge profiles facility has been incorporated to the Digital Library in order for the users to have the option to select the resource types, rates and visits thresholds etc to be used when presenting resources.

**G-REQ4.1-14**  
**Requirement**  
Administrators should have the option to update or enrich the collection of knowledge offered by the portal.

**Architecture / Analysis**  
The user interface portal provides a separate interface for Knowledge administrators who are the power users when the insertion or modification of knowledge is the subject of the administrative operations.

**G-REQ4.1-15**  
**Requirement**  
Administrators should have the option to update or enrich the collection of training material offered by the portal.

**Architecture / Analysis**  
Knowledge administrators have also the option to alter or enrich the collection of training material offered by the portal.

**P-REQ4.1-1**  
**Requirement**  
The ACCESSIBLE platform should work fine also when the developers are connected with a low bandwidth network. It is important that the system promptly communicates changes to accessibility tools, guidelines, methodologies, etc. Therefore the main aspect is to have limited time response from the system even if the connection speed is slow and even if the number of the user connected is large.

**Architecture / Analysis**  
The architecture of accessible and more specifically the usage of multiple layers for the deployment of the portal allows the portal to be operated without delays even if a low bandwidth connection is used.

**O-REQ4.1-1**
<table>
<thead>
<tr>
<th>Requirement</th>
<th>When electronic forms are designed to be completed on-line, the form shall allow people using supporting technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture / Analysis</td>
<td>The usage of EAGER makes sure that all forms that require form filling are not only usable using supporting technology but can be in turn be operated without the existence of such technology via the usage of incorporated to the framework tools such as TTS, virtual keyboards etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>O-REQ4.1-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>A method shall be provided that permits users to skip repetitive navigation links.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The usage of EAGER as the UI toolkit for the implementation of the portal allows the incorporation of several facilities such as quick access links, section breaks, skip links etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>O-REQ4.1-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>When a timed response is required, the user shall be alerted and given sufficient time to indicate more time is required.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The user interface design has been conducted with respect to this requirement.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>O-REQ4.1-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The access to the system must be realized in a one step procedure.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>Critical functions have been designed as a one step function allowing the user to perform less administrative tasks on his evaluation session that evaluation tasks.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>O-REQ4.1-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The platform must provide all the information to the end-user in a small number of steps.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The user interface design has been conducted in a way aiming at reducing the number of steps needed for performing an operation or accessing information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>R-REQ4.1-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The User Interface when combined with the appropriate hardware architecture and infrastructure, multi-tier architectures can provide the means for faultless and continuous operation.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>Multi tier setups when deployed in web farms with the usage of load balancing or clustering can offer fault tolerance that meets 99.9%.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>R-REQ4.1-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Multi-layer architecture of the User Interface will make feasible the generation of software that is scalable and maintainable allowing the separation of application logic from the User Interface and application data.</td>
</tr>
</tbody>
</table>
The existence of a multi-tier architecture allows each of the layers to expand exponentially without this growth to affect the implementation details or the communication and collaboration of these layers.

### Heading: R-REQ4.1-2

| Requirement | The architecture of the user interface portal consisting of three distinct layers who are loosely coupled enables the creation of different interface layers for each target platform. At the same time the usage of web services allows the separation of the technology used for creating those services to the specific database implementation offering support for deploying the portal on any database server by altering only the DB implementation. |

### 2.6.5 Stand alone interface requirements

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ4.2-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The User Interface of the stand alone tool should be both powerful and intuitive to use in order to facilitate both novice and experienced users.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The user interface design of the stand alone tool has been conducted in such a manner in order to be operated in the same way to a number of commercially available tools (mainly developer oriented tools). To this end such an approach offers the benefit of quickly adapting the user's conceptual model to the one propagated by the tool.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ4.2-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The user interface of the stand alone tool should provide appropriate organisation of evaluation sessions in order to be able to facilitate both the evaluation of single documents and collections of documents such as web sites residing on the local computer, collection of web services etc.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The ACCESSIBLE stand alone tool provides a specific project type that can organize evaluation sessions in the same way that development activities are organised into project by modern IDEs.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ4.2-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Users should have the option to save the specific details of each of their evaluation sessions.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The specific project type employed by the stand alone tool not only maintains a record of the evaluated documents but also acts as a storage for the specific settings selected by a user for a specific evaluation session.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ4.2-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Users and especially developers should be provided with functionality that meets the one provided by other popular developer oriented tools.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The ACCESSIBLE stand alone tool can be used as an IDE where:</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>• documents to be evaluated are organised in a project</td>
</tr>
<tr>
<td></td>
<td>• when the project is build specific warnings and errors appear</td>
</tr>
<tr>
<td></td>
<td>on the console</td>
</tr>
<tr>
<td></td>
<td>• Documents source and appearance can be viewed using the</td>
</tr>
<tr>
<td></td>
<td>incorporated editor and document browsers</td>
</tr>
</tbody>
</table>

**Heading** | **G-REQ4.2-5**
---|---
**Requirement** | The user interface shall provide the option to preview the source code of the evaluated applications.

**Architecture / Analysis** | An editor with syntax highlighting has been incorporated to the stand alone tool’s architecture for facilitating source code access. For each of the supported languages (XML, SDL, WSDL, HTML) specific syntax bindings have been defined in order to produce the syntax highlighting scheme most appropriate for each language.

**Heading** | **G-REQ4.2-6**
---|---
**Requirement** | The user interface shall provide the option to preview the evaluated documents using the most appropriate tools available according to the document format.

**Architecture / Analysis** | For each document type supported by ACCESSIBLE the stand alone tool’s architecture contains specific bindings in order for the tool to have the option to lunch the most appropriate application for previewing each document.

**Heading** | **G-REQ4.2-7**
---|---
**Requirement** | The user interface should provide users the ability to select among a number of settings in order to define the targets of each evaluation putting on the hands of the tool the matching of these setting to specific guidelines and tests.

**Architecture / Analysis** | The stand alone tool offers the project settings option. Through this option users can select among the specific disabilities, devices, functional limitations etc matching the profile of the target user population to be evaluated. Based on these settings reasoning occurs on the ontology in order for the stand alone tool to specify the specific settings to be checked against.

**Heading** | **G-REQ4.2-8**
---|---
**Requirement** | The user interface should provide users the ability to manually select among a number of guidelines the ones to be used for performing the evaluation.

**Architecture / Analysis** | From the project settings option offered by the user interface users have the option not to use the ontology and manually select specific guidelines and/or checkpoint to be evaluated.

**Heading** | **G-REQ4.2-9**
---|---
<table>
<thead>
<tr>
<th>Requirement</th>
<th>The User Interface shall provide high efficiency, reliability, and resource utilisation of the ACCESSIBLE tools.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture / Analysis</td>
<td>In the architecture of the stand alone tool the manual invocation of the jvm through .NET has been selected as the solution for performing the integration of the UI with the core architecture. This approach is considered to be the most effective among the alternatives allowing maximum efficiency reliability and resource allocation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>Requirement</th>
<th>Architecture / Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>G-REQ4.2-10</strong></td>
<td>Users should have the option to operate the stand alone tool without the need of a connection to the internet.</td>
<td>The integration approach selected requires that all the required architectural components are installed and operated on the local machine making the end product operable in a personal computer without internet connection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>Requirement</th>
<th>Architecture / Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>G-REQ4.2-11</strong></td>
<td>Users should have the option to access several different views of the evaluation results.</td>
<td>The user interface design of the stand alone tool specifies several alternative ways of viewing the evaluation results. To this end the architecture of this tool allows as to access EARL based reports as XML and preview them on a browser, generate and HTML document via the usage of xslt, or view these results as a list on the appropriate tool window.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>Requirement</th>
<th>Architecture / Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P-REQ4.2-1</strong></td>
<td>The ACCESSIBLE platform should be able to work fine also for low end computers. Therefore the main aspect is to have limited time response from the system.</td>
<td>The technologies used on the ACCESSIBLE architecture (Java and .NET) and specific optimisations conducted during development makes the system operable through the vast majority of personal computers.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>Requirement</th>
<th>Architecture / Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P-REQ4.2-2</strong></td>
<td>For time consuming operations the user should be provided with input regarding the progress of this operation and the amount of time needed to completion</td>
<td>Specific actions have been taken during the design of the interface in order for the system to keep users constantly informed about what task is currently carried out, its status, and the time needed for completion.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P-REQ4.2-3</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>For repetitive execution of similar operations no recalculation of data should occur.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture / Analysis</td>
<td>The accessible architecture specifies and integration technique where caching can be employed in both ends of the communication channel.</td>
</tr>
</tbody>
</table>

**Heading** | **P-REQ4.2-4**
---|---
| **Requirement** | Users should always have the option to quit an ongoing operation. |
| **Architecture / Analysis** | The usage of threading for the integration of the user interface with the CORE architecture of accessible makes feasible for the user interface to run on a separate thread. Therefore together with a user interface design that allows users to quit ongoing operations it becomes for the user possible to quit operations without that to result to a interface that halts. |

**Heading** | **O-REQ4.2-1**
---|---
| **Requirement** | When a timed response is required, the user shall be alerted and given sufficient time to indicate more time is required. |
| **Architecture / Analysis** | The user interface design of the stand alone tool has been conducting taking into account this requirement. |

**Heading** | **O-REQ4.2-2**
---|---
| **Requirement** | All the tasks performed by the system should require the minimum possible number of steps. |
| **Architecture / Analysis** | The user interface design has been conducted in a manner to reduce the amount of user interaction required for each action. To this end the usage of a modern UI framework for the development (.NET) enable the usage of powerful tools such as context menus, main menus, toolbars, key shortcuts in order to provide maximum interaction power with minimum user input. |

**Heading** | **O-REQ4.2-3**
---|---
| **Requirement** | When feasible alternative ways for performing an operation should be provided. |
| **Architecture / Analysis** | The architecture proposed for the stand alone tool allows us through the usage of several entry points for an action (context menus, main menus, toolbars, key shortcuts) to perform the same operation via a number of alternative ways. |

**Heading** | **O-REQ4.2-4**
---|---
| **Requirement** | User should have the maximum flexibility on customising his work space according to his requirements. |
| **Architecture / Analysis** | In the architecture for the stand alone tool dockable containers have been incorporated in order to allow end users to customise the user interface in a number of ways. Each tool-window or document can be docked in a number of positions (top, middle, left, right, and bottom) and each of these positions defines the same positions for the space it occupies. It is therefore apparent that such an interface structure offers the option to be set up in an unlimited number of variations. |
User should have the maximum flexibility on viewing the documents contained in his work space. 

The embedded in the stand alone tool editor offers the option of viewing opened documents in a number of alternative ways (split view horizontal and split view vertical). Therefore a number of documents can be viewed simultaneously.

The user interface should be stable and forgiving to user error. The architecture proposed for the use interface is highly modular allowing each of the components to be developed and tested independently. In that way the possibility of failure is reduced allowing the creation of a more stable application.

The user interface should be tolerant to errors. Specific actions should be taken during development in order to produce a application capable of recovering from system or user errors.

When a time consuming operation is carried out the interface should still remain active. The usage of threading in the architecture of accessible allows the interface to be run separately from any time consuming operations.

The user interface should be able to establish seamless communication and collaboration with the ACCESSIBLE core evaluation components. The integration strategy proposed by accessible makes the establishment of such a communication feasible.

The user interface of the stand alone tool should be loosely coupled with the ACCESSIBLE core architecture allowing the possibility of change in the lower architectural layers. The specification of a concrete communication protocol between the user interface and the core functionality of ACCESSIBLE allows the user interface not to be sensitive to changes occurring in the core ACCESSIBLE functionality.

2.6.6 ACCESSIBLE Developer/designer–aid Module Requirements

The user interface should be able to establish seamless communication and collaboration with the ACCESSIBLE core evaluation components. The integration strategy proposed by accessible makes the establishment of such a communication feasible.
<table>
<thead>
<tr>
<th>Requirement</th>
<th>The module shall help the developer/designer to ensure his application is accessible and usable to impaired users.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture / Analysis</td>
<td>The objective of the Developer/designer-aid module is to reveal to the developer/designer what the application will look like to an impaired user. This is accomplished with use of the Core Simulation module which makes a real-time impairment simulation of the application possible. This will help the developer/designer to better understand the needs of the application’s target users, preventing accessibility barriers and improving the overall quality throughout the design and implementation phases of the whole development process. Vision or upper limp impaired users can be benefited.</td>
</tr>
</tbody>
</table>

**Heading** | **G-REQ5-2 - G-REQ5-3 - G-REQ5-4 - G-REQ5-6**  
**Requirement** | The module will provide the means to simulate different low vision impairments, such as loss of central or peripheral vision. - The module will provide the means to simulate the most common low vision impairments, such as cataract or glaucoma. - The module will provide the means to simulate the most common colour-blindness impairments, such as protanomaly or deuteronomaly. - The module will provide the means to simulate different upper limp impairments, such as Parkinson disease.  
**Architecture / Analysis** | The aid module would be able to provide information for various impairments to the developer/designer. Most common low vision, colour-blindness and upper limp impairments will be implemented in order to cover as much as possible potential users of the developed application. Specifically, at least cataract, glaucoma, protanomaly, deuteronomaly and Parkinson disease will be implemented. The information regarding each impairment and the factors that affect its severity will be stored in the Accessible Ontology and will be presented to the user through the Impairments Module. |

**Heading** | **G-REQ5-5**  
**Requirement** | The module shall be delivered in two versions. One of them shall be developed as a plug-in that can be included into the Netbeans IDE and the other as a standalone application.  
**Architecture / Analysis** | The standalone version of the module will connect to the Application module so that the developer/designer can test a GUI application written in java already bundled in a jar file from another developer. The Netbeans plugin version will connect directly to Netbeans IDE. It will be able to handle both GUI forms during the development phase and GUI applications during execution phase. These two version will assist the developer/designer test third party and create java GUI applications. |

**Heading** | **P-REQ5-1**  
**Requirement** | The module should be able to simulate the desired impairment within a matter of seconds. |
### Architecture / Analysis

The Simulation Engine utilizes the JAAPI to have access to the running application. All events invoked by the application are caught and trigger a new call to the Engine. If the designer/developer wants to simulate a single GUI form then the JAAPI is not used as the Engine uses the Netbeans IDE to have access to the GUI form. This approach will make a real-time simulation possible.

<table>
<thead>
<tr>
<th>Heading</th>
<th>O-REQ5-1 - O-REQ5-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The plugin version of the module should aid the developer/designer during the development phase to handle at least Swing GUI applications. The module should be able to handle static and dynamically changing GUIs and simulate any impairment interactively.</td>
</tr>
</tbody>
</table>

### Architecture / Analysis

The plugin version will connect directly to Netbeans IDE. It will be able to handle both GUI forms during the development phase and GUI applications during execution phase. The developer/designer will be able to use a modified version of the “Preview Form” action offered through the IDE in order to simulate the developed GUI form. This way the accessibility of the GUI form will be assessed during the development phase. The developer/designer will have the chance to simulate the finished swing GUI application by actually running it with the aid of a modified version of the “Run Project” action offered through the Netbeans IDE or with the aid of the Application Module offered through the standalone version of the module.

<table>
<thead>
<tr>
<th>Heading</th>
<th>O-REQ5-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The module should give the user the ability to select any of the supported impairments.</td>
</tr>
</tbody>
</table>

### Architecture / Analysis

The developer/designer will be able to use the Impairment Chooser in order to specify the desired impairment. The set of the provided impairments are pulled from the Accessible Ontology.

<table>
<thead>
<tr>
<th>Heading</th>
<th>O-REQ5-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The user should be able to control any factors, such as color, size or severity level, that affect the selected impairment.</td>
</tr>
</tbody>
</table>

### Architecture / Analysis

Once an impairment is chosen from the Impairment Chooser the corresponding Impairments Control panel is activated. This panel contains controls for the various factors that affect the chosen impairment. These controls can be altered at will by the developer/designer in order to simulate different degrees of severity for a specific impairment. These factors are also pulled from the Accessible Ontology.

<p>| Heading | O-REQ5-4 - O-REQ5-5 |</p>
<table>
<thead>
<tr>
<th>Requirement</th>
<th>The module should be able to simulate one impairment at a time. - The module should be able to simulate a combination of two or more user selected impairments.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture / Analysis</td>
<td>The Simulation Engine takes as input the GUI application or form in question, the user specified impairment from the Impairment Chooser and the values for each control that affect the impairment from the Impairment Controls. It then produces the simulated view of the GUI. In order to simulate one or a combination of more impairments at a time a choice will be made available to the developer prior to the simulation process.</td>
</tr>
</tbody>
</table>

### Heading M-REQ5-1 - M-REQ5-4

<table>
<thead>
<tr>
<th>Requirement</th>
<th>It should be easy to add a new impairment or update an existing one to the module. - The module should be taking care of visual aspects and be highly interactive, in order to pursue users comfort ability. In addition the tool shall be configurable and provide enough documentation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture / Analysis</td>
<td>The process of adding a new impairment or updating a new one is a matter of making the appropriate actions to the Accessible Ontology, i.e. adding or updating the information regarding an impairment. Additionally, the GUI for presenting the new or updated information in the Impairments Module will be accordingly modified. All the components of the module will be written with the Java programming language making it easy to produce a high quality graphical user interface which will be easy to handle and will provide adequate information to the developer/designer when needed.</td>
</tr>
</tbody>
</table>

### Heading M-REQ5-2 - M-REQ5-3

<table>
<thead>
<tr>
<th>Requirement</th>
<th>The module shall be able to run in any JavaSE enabled system (with JavaSE 6.0 or above). - The module shall be able to run in any operating system that supports JavaSE, including Windows, Macintosh, OpenSolaris, or Linux.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture / Analysis</td>
<td>The Developer/designer-aid module, the Application module, the communication with the Accessible Ontology and the Netbeans plugins will be written in the Java programming language. Thus, it will be straightforward to be run in any Java enabled operating system.</td>
</tr>
</tbody>
</table>

### 2.6.7 ACCESSIBLE EARL-based reporting tool Requirements

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ6-1 (G-REQ6-8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The EARL-based reporting tool shall be used in order to present the results of the Accessibility Evaluation in a standardized and easy-to-understand way.</td>
</tr>
</tbody>
</table>
The Accessible EARL Report Generator incorporated within the Accessible EARL-based reporting tool will have the ability to transform the Accessibility Assessment Result produced from the Accessible Assessment Simulation Module into a report containing both machine-readable values and human-readable and easy-to-understand descriptions regarding the assessment results. Moreover, the presentation of the results will have as a basis the EARL standardization, ensuring the standardized presentation of the assessment’s outcome.

### G-REQ6-2 (G-REQ6-3, G-REQ6-4, G-REQ6-5, G-REQ6-6)

**Requirement**

The EARL-based reporting tool will incorporate terms and constraints from the W3C Evaluation and Report Language.

**Architecture / Analysis**

The W3C Evaluation and Report Language proposes a group of labels to identify the tool, the person who is performing the analysis, the tested content and also, assertions which contain the result of the test performed. Building upon this specification, the Accessible EARL Based Reporting Tool will produce Assessment reports containing information regarding the person who or the tool that carried out the test (G-REQ6-3), the subject that was evaluated (G-REQ6-4), the test criteria against which the subject was evaluated (G-REQ6-5) and the outcome of the test conducted on the subject (G-REQ6-6).

For each assessment conducted, all this information will be inputted into the Accessible EARL Report Generator from the Accessible Assessment Simulation Module and the Accessible User Interface Portal. The result will be the generation of an EARL-based report containing the complete assessment’s description.

### G-REQ6-7

**Requirement**

The test result can contain information such as the success or failure of the test, a confidence indicator for the obtained results, the reason for failing, or other attributes.

**Architecture / Analysis**

The different types of tests implemented within the Assessment Simulation Module may have different types of results, ranging from simple Boolean values (pass – not pass) to statistical confidence indicators. Furthermore, in some cases additional information may be needed within the test’s outcome in order for it to be fully described. The EARL Report Generator incorporated within the Accessible EARL-based Reporting Tool will take this into account, allowing for this types of information to be acquired from the Assessment Simulation Module and be presented within the reports generated from the tool.

### G-REQ6-8

**Requirement**

The test result shall include both machine-readable values as well as human-readable description of the results.
Architecture / Analysis

The machine-readable values will be the absolute indicators of each evaluation outcome. These will be used from the Accessible User Interface Portal in order to present the results in a user-friendly way. Furthermore, by incorporating human-readable descriptions of the results within the produced reports, the tool will allow for system users (accessibility assessment experts) to better understand and analyze the assessment’s outcome, by viewing only the generated EARL-based report.

<table>
<thead>
<tr>
<th>Heading</th>
<th>G-REQ6-9 (G-REQ6-10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Any extensions to the EARL vocabulary must ensure the integrity and validity of the core EARL vocabulary</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>The reports generated from the tool will build upon the structure defined within the EARL specification. Thus, the EARL Report Generator incorporated within the tool will have to use the vocabulary already defined within the EARL specification. However, due to the fact that the reports will have to present descriptions of different types of tests, there might be the need to define new elements regarding the reports’ structure, possibly by extending the vocabulary already defined within the EARL specification. For instance, extensions will be probably needed in order to cope with the needs of the Web accessibility evaluation domain. These extensions to the already defined EARL vocabulary, if made, should ensure no conflicts to the core EARL vocabulary, in order for the integrity and validity of the latter to be conserved.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading</th>
<th>P-REQ6-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Generated reports shall be user oriented. For example reports for Web developers may contain more detailed results in order to repair bugs while reports generated for project managers may contain aggregated results with less detail on the specific issues.</td>
</tr>
<tr>
<td>Architecture / Analysis</td>
<td>Information regarding the type of the user performing the assessment will be acquired (if available) from the Accessible User Interface portal (in the Web Accessible case). This information will be used from the Accessible EARL Report Generator, in order to decide the level of detail contained within the final report. For example, if the user has been registered as a Web developer, the report generated for her/him may contain more detailed results in order to repair bugs etc. On the other hand, reports generated for users registered as project managers may contain aggregated results with less detail on the specific issues, allowing them to more easily understand the evaluation outcome. Furthermore, the production of less complex reports where appropriate, will allow in some cases the EARL Report Generator to consume less system resources than the ones required for the generation of a Web Developer-oriented report.</td>
</tr>
</tbody>
</table>
3 ACCESSIBLE Conceptual Architecture

3.1 ACCESSIBLE Architectural scheme

Designing for people with disabilities is becoming an important topic today. This idea is strongly supported by the fact that an increasing number of countries are legislating towards promoting and enforcing the rights of people with disabilities. Consequently, accessibility is one more aspect that has to be taken into account in the development of software applications, especially in user interfaces.

Furthermore, accessibility can be perceived in different angles. What is accessible to one person, might not be accessible to another one. The different requirements to access and interact within applications pose a significant challenge on how these should be developed. This task often includes the development of different user interfaces to support a particular user group, as well as an augmentation of user interfaces with accessibility concerns that can be interpreted by assistive technologies.

However, the development of accessible software requires a strong effort from developers, testers, etc. With the additional encumbrance of taking into account different kinds of accessibility requirements, guidelines and best practices, and different user interface implementation technologies (which by themselves might pose severe problems of delivering accessible applications), developers are faced with a daunting task.

Therefore, the highly specialised skills required for developing accessible software sets aside most developers. To mitigate these problems, developers should be guided in their development process about accessibility concerns within software development. This includes the definition of target users (e.g., their requirements, disabilities, etc.), which aspects should be taken into account to meet users’ accessibility expectations, and how it reflects on software applications (thus coping with the particularities of different technologies). To overcome the gap between developers knowledge on accessibility issues and the development of accessible and tailored software applications, ACCESSIBLE project will provide the potential users with an overall assessment and developer designer aid framework for the development of accessible software applications.

The overall layered architecture of the proposed system is shown in the following Figure 9. The proposed architecture specifically addresses concerns about automation over accessibility testing and it compromises with independent modules that can interact each other. Towards this goal, the ACCESSIBLE working environment develops a modular framework that provides the interface between users and the following interacting components:

- ACCESSIBLE Portal/ User Interface of the system
- The assessment simulation modules,
- The ACCESSIBLE ontological Knowledge Resource
- The developer/designer–aid module and finally the
generated analysis reporting tool.
3.1.1 ACCESSIBLE User Interface

Different architectural approaches are relevant for the ACCESSIBLE User interface architecture as identified below:

- An overall Web User Interface portal that can support users that they would prefer to verify the accessibility of their applications online with the adoption of relevant Web Services
- Different Standalone User Interfaces for the Open source standalone applications (where appropriate) that can be downloaded to users terminals and can be used for the accessibility assessment of their software components

In terms of appearance, the ACCESSIBLE Web user interface can be thought of as a hyper-portal where relevant users can use the Web based services of the ACCESSIBLE components, to extract useful information for accessibility guidelines,
standards, etc an finally to download the standalone modules of the ACCESSIBLE project. That said access will take place using a common Internet browser. The user-centred presentation portal will be dedicated to specific skilled users (developers, programmers, testers) as well as to non specific that enable them both to design and develop accessible software applications and also to evaluate the accessibility of their software applications in order to get accessibility-status reports through a variety of tools and innovative technologies.

Regarding the potential structure and functionality of the Portal, different UI presentations according to specific user groups will be supported, enabling a personalised experience in relation to specific tasks. More specifically, each different user group, according to the requirements stemming by its role in the development process, will have access to different facilities and methods, and furthermore the ability to manage and alter these dynamically populated collections. Typical facilities offered include the personalisation of the user work space, project-based presentation of accessibility evaluation sessions, repositories of methods and tools according to their popularity or personal preference, unified project workspace for testing and reporting, etc. Also, relevant users of the system can have the possibility of selecting the more appropriate accessibility method or tool they prefer based on their exact requirements and preferences. The portal interacts with all the main components of the ACCESSIBLE system such as the assessment simulation module, the multilayer ontology, the designer aid module and the EARL reporting tool. The design of the user-centred presentation portal will be carried out using the user centred design approach, which is an iterative process whose goal is the development of usable systems, through the involvement of end users in the design process.

The architecture of the portal is based on an advanced configuration, called three- or multi-tier architecture, aiming at separating the application from logic and data. The proposed architecture introduces several advantages such as scalability, reliability and security. Furthermore this complete distinction of the application tiers from the data tiers enables the integration of various services for unifying the ACCESSIBLE user interface with the Simulator Assessment Module, Developer and Design-aid Module and the EARL based Reporting Tool.

### 3.1.2 ACCESSIBLE Assessment Simulation Modules

The Assessment Simulation module will support the overall analysis and verification in terms of accessibility for Web applications, Web services mobile applications and description languages (UML, SDL, etc.). The module, which takes input from the ACCESSIBLE knowledge resource, will be composed of independent accessibility assessment tools in order to support the overall accessibility assessment process. This module includes:

- **A Web applications assessment tool** (Web and Standalone version) for the accessibility verification of Web applications. The available Structure Evaluation Tool (provided by NETSCOUTS) will be integrated to this module and will be used for the visualisation of the assessment results. The Structure Evaluation Tool is a software tool for the analysis and validation of the structure of websites and applications, in terms of accessibility and quality (more information can be extracted to Annex I). In addition SET can check failure detection capabilities, based on coloured
highlighting (such as sequence of headlines, missing attributes in titles or graphics, control of table layouts, language break, etc.).

- A **Web services assessment tool** for the accessibility verification of Web services, which will be based on the open source web-based service alignment tool (provided by CERTH/ITI from ASK-IT integrated project) in order to support the assessment in terms of accessibility for web services. (more information can be extracted to Annex I)

- A **Mobile applications Assessment Module** for the accessibility verification of mobile applications

- A **description languages assessment tool** for the accessibility verification of description languages. Concerning description languages the tool will offer the possibility to check the compliance of the designs with accessibility guidelines as described in the ACCESSIBLE harmonised methodology (ACCESSIBLE ontological knowledge resource). This module is based on the available SAFIRE Professional tool, which is a fully integrated development & run-time environment optimized for the implementation, validation & observation of telecoms systems developed and marketed by SOLINET (more information can be extracted to Annex I).

### 3.1.3 ACCESSIBLE ontological knowledge Resource

To overcome the gap between developer’s knowledge on accessibility issues and the development of accessible and tailored software applications, an multilayer ontological knowledge resource framework is presented by ACCESSIBLE project. The main purpose of this framework is the division between generic and domain-specific accessibility concepts, and how they can be mapped within accessibility assessment scenarios. Hence, ACCESSIBLE ontological knowledge resource aims at providing support for the formal and unambiguous definition of accessibility domains, as well as the possible semantic interactions between them. We have specified generic and domain ontologies to be integrated into accessibility verification environments. This will establish a common vocabulary for exchanging and describing the complex information that is related to accessibility assessment of software applications and, ultimately, guide developers on the creation of accessible user interfaces.

This framework will incorporate the ACCESSIBLE harmonized methodology which has been presented in deliverable D3.1 [25] and aims to formalize conceptual information about:

- The characteristics of users with disabilities, assisted devices, applications, and other aspects that should be taken into account when describing an audience with disabilities and developing tailored Web sites
- accessibility standards and associated checkpoints and guidelines
- Semantic verification rules (through appropriate SWRL (Semantic Web Rule Language) and SPARQL rules and queries) to help describing requirements and constraints of users, and associating them to accessibility checkpoints

Therefore, one of the main issues in designing the proposed framework is to make it maintainable and extensible, while assuring consistency of accessibility assessment
scenarios. In order to cope with these goals, the architectural framework will comply with the following requirements:

- To be as *formal* as possible, thus providing all the necessary definitions in a concise, unambiguous, and unified form
- Provide information that can be *easily processed* by software applications and integrated into accessibility assessment processes

Towards this goal, the ACCESSIBLE ontological knowledge resource framework develops a modular data framework that provides the data infrastructure for the ACCESSIBLE assessment modules as well as the developer designer aid module.

### 3.1.4 ACCESSIBLE Developer and designer aid module

It is usually difficult for designers and developers to understand the problems users with disabilities face when accessing their software implementations that are not designed with their needs in mind. Thus, in order to have a better view of accessibility needs, a relevant developer and designer aid module is needed in order to provide an opportunity for users to experience a software application (e.g. Web page) using simulated disabilities.

The purpose of module is to assist developers and designers to better empathise with those who have reduced capabilities, and to help understand how capability loss affects the ability to interact with software applications and services. The philosophy behind the implementation of this module is to provide a complete free and open Source software application toolkit that can be offered through two versions, a standalone version and a plug-in version for the NetBeans IDE. These tools can enable, encourage, and assist users ("authors") in the creation of accessible applications. This tool will be centred on the ACCESSIBLE harmonized methodology and ontological framework, thus ensuring accessibility quality. Target users (developers, designers, etc.) will be supported with specific tools that afford designing and implementing accessible software applications. These tools will be based on already existing open-source software architectures and technologies, such as the NetBeans IDE, JA API, etc.

### 3.1.5 ACCESSIBLE Generated Analysis Reporting Tool

The Evaluation and Reporting Language (EARL)-based reporting tool in order to export accessibility evaluation results in a form helpful to potential receivers of test results, include designers, developers and business stakeholders. The Evaluation and Reporting Language (EARL), which has been proposed by W3C evaluation and Repair Tools Working Group (ERT WG) [26], is a language to express test results such as bug reports and conformance claims. EARL enables any person, entity, or organization to state test results for any thing tested against any set of criteria. The needs of audience are likely to be somewhat different. For example a developer is primarily interested in a page-by-page enumeration of errors and recommendations, whereas a manager is more concerned with compliance issues and strategic recommendations. The following four core classes from EARL will be incorporated within ACCESSIBLE reporting tool:

- **Assertor**: Context information about the test and contains information such as who (person) or what (tool) carried out the test, when the test
was carried out, or information about the platform and configuration in which the test was executed.
- Subject: The subject which is being evaluated and could be a piece of software code, a document, or an object.
- Testcase: The test criteria against which a subject is being evaluated could be any (referenceable) specification, guidelines, a single test from a test suite, or some other test case.
- Result: The outcomes of a test conducted on a subject can contain information about the success or failure of the test, a confidence indicator for the obtained results, the reason for failing, or other attributes.

3.2 ACCESSIBLE subsystems Logical architecture

3.2.1 ACCESSIBLE Assessment Simulation Module

As described to previous sections the Assessment Simulation module will provide support to users for the overall analysis and accessibility verification, of software components such as Web and mobile applications, web services and description languages.

Thus, the assessment module will be composed by the following independent components:

- A web applications assessment tool
- A mobile applications assessment tool
- A Web services assessment tool
- A Description languages assessment tool

The logical architecture of the above assessment tools is presented to the following sections.

3.2.1.1 The Web applications assessment tool

The ACCESSIBLE Web Applications Assessment tool will be responsible for the accessibility evaluation of Web Applications. For the convenience of ACCESSIBLE users two different versions of the tool will be supported:

- A Standalone version, which will be offered for downloading from the Accessible Web Portal. The user after downloading and installing the specific application in his/her personal computer will be able to evaluate a web page or an entire web site that is published on the internet or is stored locally in its computer. The standalone version can take as input the URL/local path of the preferable web page (or the URL/local path of the home page if an entire web site is going to be evaluated) and then it can produce as output the evaluation results of the accessibility assessment process. Each user will also be able to define some input parameters/rules regarding the assessment process (ex: evaluate a web page according to a specific accessibility standard or according to a specific disability, etc.). As depicted in the architectural scheme of the following Figure 10, the Web
Applications Assessment standalone tool consists of five main components as described below:

1. **A Graphical User Interface (GUI):** The GUI of the Web Applications Assessment tool consists of a set of relevant usable forms that can be accessed by users in order to help them with the accessibility assessment of their software components. Also, the GUI of the assessment tool is responsible for the representation of the assessment results during the evaluation process. Based on the functionalities of this tool, the user can insert the preferable URL/local path of the web page/site that is going to be evaluated. If an entire web site is going to be evaluated, the user can have the possibility to insert the preferable relevant number of the web pages that would like to be examined. Additionally, the user can select the preferable approaches to be executed using the stored knowledge of the Accessible Ontologies. The Rules Inference Engine can pass all the necessary inferenced knowledge to the GUI, in order to give users the opportunity to make specific selections regarding the assessment process. The ACCESSIBLE knowledge resource such as the accessibility standards, impairments, disabilities, functional limitations, rules, etc., that are included within the ACCESSIBLE ontologies can be presented and accessed through the GUI in order to make the preferable selections. The GUI can also provide a manual way for the selection of the proposed accessibility approaches (e.g., accessibility rules, standards, etc.) that can be executed by the Web Accessibility Evaluator. For instance, all the disabilities described in the ontology will be shown in a list and the user may choose the disabilities according to which the evaluation process will be performed. When the user will have made the desired selections, the GUI will pass to the Web Accessibility Evaluator the URL/local path of the web page/site as well as the selected approaches and the evaluation process can start. After the finalisation of the evaluation process, the Web Accessibility Evaluator will return the results to the GUI so as to be presented to the user.

2. **SET Tool:** In order to achieve a better visualization of the results of the assessment process, the GUI of the Web Applications Assessment tool integrates the SET Tool (annex I) which has been developed by NETSCOUTS.

3. **The Rules Inference Engine:** The Web Applications Assessment tool has to be connected with the AccessibleOntologies in order to use the stored knowledge concerning accessibility standards and their guidelines, user disabilities and functional limitations, assistive devices, etc. This connection will be achieved by the Rules Inference Engine module. The Rules Inference Engine will be able to run all the SWRL rules defined in the ontology and provide all the inferenced knowledge to the Web Applications Assessment module. The Rules Inference Engine will also be able to execute SPARQL queries, in order to extract specific knowledge from the ontology. The Rules Inference Engine will communicate with the GUI of the Web Applications Assessment tool which will present the necessary inferenced knowledge (e.g., the accessibility standards/disabilities/functional limitations/impairments and their
correlations) to the user in order to make the desired selections regarding the evaluation process.

![Logical architectural scheme of the Web Applications Assessment standalone tool](image)

**Figure 10:** Logical architectural scheme of the Web Applications Assessment standalone tool

4. **The XML Storing/Loading Module:** Because in many cases the execution of all SWRL rules which are defined in the ACCESSIBLE ontology knowledge resource can be considered as a huge time-consuming process, an XML storing/loading module is being introduced within the Web Applications Assessment Module. Thus this module will be able to automatically generate an XML file containing all the necessary knowledge of the ontology that is required for the assessment process. After the generation of the XML document the XML Storing/Loading module will be responsible for the “virtual connection” between the Web Applications Assessment tool and the ontology. It becomes clear that the Rules Inference Engine module should be used every time the Accessible ontology is being updated. Then the XML Storing/Loading module will be able to produce the updated XML file, which will be used in further executions of the Web Applications Assessment tool.

5. **The Core Web Applications Assessment Module:** The Web Applications Assessment module is the core component of the Web Applications Assessment tool and includes all the required algorithms
and methodologies for the execution of preferable user’s selections. Thus, this module consists of the following sub-components:

I. The Web Crawler
The Web Crawler can be used when an entire web site is being selected for its accessibility evaluation. The crawler can take as input the URL homepage of the selected web site and then it can browse automatically all the pages included in this web site. The URLs of the pages will be stored in a list that will be passed to the Web Accessibility Evaluator in order to evaluate each web page separately and present the assessment results. The preferable number of the web pages that should be examined can be defined by the user in the GUI of the Web Applications Assessment tool.

II. The W3C Markup Validator (requires access to the internet)
Most Web documents are written using markup languages, such as HTML or XHTML. These languages are defined by technical specifications, which usually include a machine-readable formal grammar (and vocabulary). The act of checking a document against these constraints is called validation, and this is what the W3C Markup Validator [27] does. The Web applications assessment tool integrates the W3C Markup Validator in order to return to users all the potential errors and warnings that can be found in the examined web pages. The W3C Markup Validator is an open-source script written in Perl and for its proper execution it has to be installed in a web server that also contains all the necessary modules for Perl language. Due to the restriction of the server existence, the W3C Markup Validator cannot be integrated in the Web Applications Assessment standalone tool. Instead, it will be installed in the Accessible Web Server and the communication with the Web Applications Assessment tool will be performed over the internet. If the personal computer of the user does not have access to the internet, the communication with the W3C Markup Validator will not be feasible and finally the Web Accessibility Evaluator will produce results that will not include the HTML/XHTML errors and warnings for the evaluated web page/site. In this case, the functionality of the Web Applications Assessment standalone tool will be limited but the loss of connection with the W3C Markup Validator will not be a barrier for the proper execution of the tool.

III. The HTML Parser
The HTML Parser can use Groovy scripts in order to parse the web page source code and get the necessary information concerning the desired elements/attributes of the HTML/XHTML. Groovy is a powerful high level language for the Java platform which compiles down to Java bytecode.

IV. The W3C CSS Validator
The W3C CSS Validator Service is a free software created by the W3C organisation to help Web designers and web developers to check Cascading Style Sheets (CSS). It takes as input a CSS file, compares it to
the CSS specifications and finally returns all the errors and the warnings found. According to the CSS 2.1 Specification [28]: The validity of a style sheet depends on the level of CSS used for the style sheet. A valid CSS 2.1 style sheet must be written according to the grammar of CSS 2.1. Furthermore, it must contain only at-rules, property names, and property values defined in this specification. By default, W3C CSS Validator checks style sheets against the grammar, properties and values defined in the CSS 2.1 specification, but other CSS profiles can be checked against by using the available options. The W3C CSS Validator will be integrated in the Web Applications Assessment module. The Web Accessibility Evaluator will send to the W3C CSS Validator the URL of the CSS file to be examined and the W3C CSS Validator will return to the Web Accessibility Evaluator the errors and the warnings that have been found in the CSS.

V. The CSS Parser
A CSS parser will be integrated to the Web applications assessment module. The CSS Parser module will be responsible for the parsing of all the CSS files that are connected with the examined web page. For this purpose, an open-source tool, which is also called “CSS Parser”, will be integrated. The “CSS Parser” is implemented as a package of Java classes that inputs Cascading Style Sheets Level 2 source text and outputs a Document Object Model Level 2 Style tree. Alternatively, applications can use SAC: The Simple API for CSS [29]. Its purpose is to allow developers working with Java to incorporate Cascading Style Sheet information, primarily in conjunction with XML application developments. The CSS Parser module will return to the Web Accessibility Evaluator all the desired CSS rules and properties.

VI. The Web Accessibility Evaluator
The Web Accessibility Evaluator is the core component of the Web Applications Assessment module. It takes as input the URL/local path of the web page/site that needs to be examined and evaluates the web page/site as far as it concerns the accessibility rules according the preferences of the user. Initially, the user via the graphical user interface (GUI) of the tool can select the relevant approaches (accessibility checks) that the Web Accessibility Evaluator can execute either manually one-by-one (ex: using checkboxes) or using the knowledge of the ontology (ex: the user may check the accessibility of the page for impaired users having colour blindness and using WCAG 2.0). In the second case, the GUI will co-operate with the Rules Inference Engine or the XML Storing/Loading module.

The Web Accessibility Evaluator in collaboration with the integrated W3C Markup Validator and the W3C CSS Validator can firstly support the users of the system to receive detailed information about possible errors in the structure of the HTML/CSS respectively and secondly to indicate possible accessibility errors and/or warnings based on appropriate approaches supported by the tool.

In case that an entire web site has to be evaluated for its accessibility, the Web Crawler of the module can return to the Web Accessibility
Evaluator all the URLs of the web pages that are included to the Web site. Then the Web Accessibility Evaluator can evaluate each page separately and present the assessment results. Relevant algorithms as defined in the deliverable D3.2 [30] will be integrated in order to calculate the assessment results for each Web site and/or Web page. Also, an XML format of the assessment results can be provided to the ACCESSIBLE EARL-based reporting tool, which is responsible for the EARL-based accessibility report generation.

- A Web Service version that can accessed through the ACCESSIBLE portal. Thus, any user will be able to use the assessment functionalities of the Web Portal in order to evaluate the preferable web page or an entire web site which is published on the internet. It is obvious that the web service version will not support the accessibility assessment of web pages that are stored locally on the computer of users. The overall functionalities of this Web version will be supported by the main components of the Web Applications Assessment Module, which has been described above and are:
  - The Web Crawler
  - The W3C Markup Validator
  - The HTML Parser
  - The W3C CSS Validator
  - The CSS Parser, and
  - The Web Accessibility Evaluator

The logical architecture of this tool is depicted to the following figure 11.

![Logical architectural scheme of the Web Applications Assessment tool (Web services version)](image-url)
Each user can insert to the portal the preferable URL of the web page/site that needs to be examined and select the relevant approaches to be executed. The selection of the approaches can be done manually one-by-one and using the knowledge stored in the ACCESSIBLE Ontologies. The Rules Inference engine is responsible for passing all the inferred knowledge of the Accessible Ontologies to the Web Interface of the assessment tool in order to be accessed by users. The assessment procedure is coordinated by the Web Accessibility Evaluator module which can calculate all the evaluation results based also on the other components of the Web Applications Assessment module that has been described before (W3C Markup validator, W3C CSS validator). For the visualization of the results the EARL-based reporting tool, which is responsible for the EARL-based accessibility report generation can be used.

### 3.2.1.2 The mobile application assessment tool

In the context of the ACCESSIBLE project, mobile evaluation has particular requirements and challenges, in comparison with other technologies, caused by the mobility aspect. Thus, we have defined the architecture for an evaluation tool with the intent of providing support to Web developers and designers on conducting rapid, yet specialized, accessibility assessments focused on any relevant disability types, target platforms and contextual conditions for Web contents targeted to mobile devices.

Each user has the possibility to select specific rules as defined to the ACCESSIBLE Ontologies (e.g. disability types) in order to set the active tests that can be performed, as defined by the Evaluation Modules. This mapping of rules is based on the ACCESSIBLE harmonised methodology detailed in Deliverables D3.1 and D3.2. Thus, the application of selected tests will generate an evaluation report detailing the problems (errors and warnings) that have to be fixed by relevant users (e.g Web developers, designers, etc.). In the following Figure 12 we present the logical architecture for the Mobile Web Accessibility evaluation tool.
As presented to the figure above, the evaluator’s architecture consists of the following components:

1) **Mobile Web Accessibility Evaluator**: This is the main component of the Mobile Web Accessibility assessment tool, where Web pages must be evaluated according to mobile Web accessibility assessment standards. This process is staged in three sequential components:
   
a. **Content negotiation/parsing**: This component in the architecture is responsible for HTTP negotiation with Web servers, in order to obtain responses that are tailored to mobile devices, when available. With this outcome, an XML parser must transform the corresponding Web page into a machine-friendly memory structure (e.g., XML tree), for subsequent evaluation;

b. **Evaluation modules selection and execution**: since one of the goals of ACCESSIBLE lays in the personalized and tailored assessment of accessibility, the Mobile Web Accessibility assessment tool must incorporate this functionality. Hence, this step integrates the outcome from running the **Rules Inference Engine** and capturing its output, in order to afford the selection of technology-specific evaluation components that have been mapped into disabilities according to the Harmonised Methodology (Deliverables D3.1 and D3.2) in the **ACCESSIBLE ontologies**. These modules are:
i. WCAG module: must implement the tests from the Web Content Accessibility Guidelines, through the integration of the Web Applications Assessment module of ACCESSIBLE;

ii. MWBP module: must implement the tests from the Mobile Web Best Practices, as detailed by the W3C Mobile Web Initiative.

With selection of the individual evaluation components, the execution step triggers the evaluation process applied to the memory structure made available by the Content negotiation/parsing component;

c. Reporting: A reporting component is responsible for taking the outcome of the execution step and made it available in different output formats (e.g., EARL).

2) User Interface: In order to be useful, the MWA evaluator must be easily manipulated/configured by its target users. Hence, a Graphical User Interface (GUI) needs to be developed, which allows triggering the selection activities (as depicted in Figure X) and capture the evaluation’s return report. This User Interface must be implemented in such a way that selection and return are available to the user. For that, and to cope with the goals of ACCESSIBLE project, the implementation must be done in two fronts:

a. Stand-alone tool: the evaluator must provide a GUI that will provide access to the evaluation features described previously.

b. ACCESSIBLE portal: the evaluator must make its features available through an Application Programming Interface (API), such as through Web Services, in order for it to be integrated into the ACCESSIBLE portal.

As for the content flow, the mobile Web accessibility assessment tool consumes and produces the following data:

1) Inputs: an HTML Web page, or a set of HTML Web pages, and associated resources (e.g., CSS, Javascript, Images, etc.), which can be properly tailored to mobile devices, if content negotiation was performed by the Web servers providing those resources;

2) Outputs: a report with the outcome of the accessibility evaluation (e.g., stating the failure/pass of specific accessibility assessment tests).

Lastly, the following technologies were considered as a starting point in the architecture of the mobile Web accessibility assessment tool, in order to ensure a successful accomplishment of the implementation of this tool:

- JVM-based technologies and libraries;
- Groovy programming language and associated libraries;
- The HTTPBuilder content negotiator;
- Parsers and manipulators of Web resources such as HTML Web pages, e.g., NekoHTML, Groovy internal XmlSlurper;
- All technology and library dependencies of the Rules Inference Engine (Deliverable 4.2).
3.2.1.3 The web services assessment tool

The ACCESSIBLE Web Service Assessment module is the one responsible for the evaluation of Web Services. This module communicates with (a) the Accessible Ontology in order to get information regarding the Web Services Accessibility Guidelines defined and (b) the EARL-based Reporting tool in order to provide it with all the necessary information for the generation of EARL-based reports regarding the Accessibility of Web Services and present them in users. If the Web services assessment tool considered as a black box, the module can take as input the relevant WSDL file which describes the Service under evaluation and the Web Service Accessibility Guidelines defined within the Accessible Ontology, and produces as output the result of the WS Accessibility Assessment process.

![Block Diagram of the Accessible Web Service Assessment Module](image)

As depicted in Figure 13 the Accessible Web Service Assessment module consists of three main components:

1. **The Accessible WSDL Parser**
   This component is responsible for the parsing of “Web Service Description Language” (WSDL) files. It takes as input the WSDL file describing a Web Service and produces as output Java structures that hold information regarding the Web Service, appropriate for further processing and accessibility evaluation. This component’s functionality is based on the Apache Axis 1 and Axis 2 W3C SOAP protocol implementations.

2. **The Service Alignment Tool**
   Initially developed for the purposes of the ASK-IT integrated project, this web (PHP) based tool offers service evaluators the capability to “align” Web Services to the Accessible WS “Ideal Operations” defined within the Accessible Ontology. All the executed alignments can be stored within an appropriate Service Alignment Database.

3. **The Accessible Web Service Evaluator**
   This component is a Java-based application which takes as input:
a) Information derived from the parsing of the WSDL file
b) Information derived from the alignment of the Service’s operations to the concepts defined within the “Ideal” ones, stored within the Accessible Service Alignment Database
c) The Web Service Accessibility Guidelines defined within the Accessible Ontologies

The Accessible WS Evaluator combines these three inputs and produces as output the WS Accessibility Assessment result, which is then passed to the Accessible EARL – based reporting tool responsible for the EARL-based Accessibility Report generation.

The assessment process supported by the functionalities of the Web Service Assessment module, which is presented to figure 14, consist of the following steps:

1. **Parsing of a Web Service’s WSDL file:** During this initial step, the Accessible WSDL Parser acquires information regarding the operations defined within the Service under evaluation. This is done through the “parsing” of the Service’s WSDL file. During the parsing process, all the information contained in the WSDL file is transferred in Java-based structures, appropriate for further processing and evaluation of the Accessible WS Accessibility Guidelines. The Accessible WSDL Parser is able to parse among others the most commonly used “rpc/encoded” and “document/literal” types of WSDL files.

2. **Automatic evaluation of the Service’s accessibility status based on information acquired from step 1:** Within this step, all information acquired from step 1 is used from the Web Service Evaluator in order to evaluate a limited set of the Service Accessibility Guidelines. This limited set contains all Accessibility Guidelines that can be automatically checked by using the information acquired so far from step 1.

3. **Alignment of the service’s operations to the Accessible “Ideal Operation” elements, defined within the Accessible Ontologies:** By utilizing the service

---

**Figure 14: Overview of the Accessible WS Evaluation process**
alignment capabilities offered from the ASK-IT Service Alignment Tool, the Accessible Web Service Assessment Module acquires more information regarding the Service’s operations and their input and output structures. Within this process, the Service evaluator is asked to align the operations defined within the Service’s WSDL file and their input and output elements to corresponding ones, defined within the Accessible “Ideal Operations”. The alignments produced are stored inside the Accessible Service Alignment DB, ready to be used from the Accessible WS Evaluator during Step 4.

4. **Automatic evaluation of the Service’s accessibility status based on the combined information acquired from steps 1 and 3:** Within this step, all information acquired from steps 1 and 3 is used in order for the Accessible WS Evaluator to evaluate a broader set of Guidelines than the one assessed during step 2.

5. **Manual evaluation of the Guidelines that cannot be assessed automatically by using the information acquired from steps 1 and 3:** During this step, the (user) evaluator of the service manually evaluates the service against the Guidelines not previously checked until step 4. The evaluator, for the purposes of this task, is offered the capability to invoke the Web Service’s operations by using the dynamic invocation option provided from the Accessible WSDL Parser, in order to check more Accessibility Guidelines (the ones that require Service Invocation in order to be assessed).

### 3.2.1.4 The description languages assessment tool

The Description Languages Assessment module provides an application capable of extracting the accessibility characteristics and features that an SDL system can have and comparing the values of the application’s features to the required feature values that an accessible application must have. By making this comparison the application provides an Accessibility Assessment Report indicating an SDL application’s accessibility level. The module also makes suggestions of how the accessibility level of the application can be increased.

An SDL application that is going to be assessed will be developed using the SAFIRE tool and its Graphical Editor which are provided by SOLINET beneficiary. Further details about SAFIRE can be found in the Annex I of this deliverable. The Graphical Editor will be used for the creation of an SDL system based on the pre-defined GUI template. A relative snapshot from the editor is depicted in the following figure 15.
The Editor also produces the SDL system in text format file that contains the application source code according to SDL 2000 standard, a part of which can be seen in the following table 3. The source code files are stored with file extensions such as “.pr” and “.fsm”.

```
INPUT SWGT_Handle_S(Camera_Handle) VIA Camera_Grp_G;
  OUTPUT Button_Create_S(Camera_Handle) VIA Btn_G;
  OUTPUT SWGT_Label_Create_S(Camera_Handle),
      VIA Mem_Lbl_G;

  NEXTSTATE -;

  INPUT SWGT_Handle_S(Mem_Lbl_Handle) VIA Mem_Lbl_G;
  OUTPUT SWGT_Label_Create_S(Camera_Handle),
      VIA Timer_Lbl_G;

  OUTPUT Create_Photo_Store_S(Camera_Handle) VIA
      Photo_Store_G;
  OUTPUT Display_Photo_S(28) VIA Photo_Store_G;
  NEXTSTATE -;
```

Table 3: SDL source code sample

The following figure 16 depicts an overview of the Description Language (SDL) Accessibility Assessment Module Architecture. The system is a software component that parses SDL system files and based on some pre-defined accessibility criteria, it calculates the accessibility level of the SDL system. Then it suggests changes that can be done in the SDL system in order to enhance the application’s accessibility level.
This textual representation of the SDL system will be used for the accessibility evaluation procedure using the SDL accessibility assessment module. Further details are provided in the following sections.

More specifically the Accessibility Assessment Module consists of the following three sub-modules:

- The **Accessibility Threshold Controller** that is responsible for connecting to the external ontology and get the guidelines and values for assessment. It also obtains the thresholds that will be used for the evaluation. Some of the defined accessibility features include:
  - Window size
  - Button size
  - Fonts size and color
  - GUI components position
  - Resolution of figures
  - Existence of error messages (graphical)
  - Existence of error messages (voice)
  - Existence of voice commands
The **Accessibility Features Parser**, which is capable of parsing the SDL file and storing the accessibility semantic features. The module first communicates with the Accessibility Threshold Controller entity to get the application features that must be checked for the accessibility evaluation. It then parses the SDL system application’s source code (source code files are these with extensions “.pr” and “.fsm”) and looks for specific SDL commands and seeks for the values of command parameters. Some of the supported parameters that can be checked for every feature are:

- Window size – Length and width parameters
- Button size – Length and width parameters
- Fonts – Font size and color parameters
- GUI components position – X and Y coordinate parameters of components
- Resolution of figures – Length and width parameters
- Existence of error messages (graphical) – Error handling and color of error and information messages.

**Figure 16: System Architecture of the Accessibility Assessment Module for SDL Systems**
- Existence of error messages (voice) – Implemented voice commands
  handlers for error messages.
- Existence of voice commands - Implemented voice commands
  handlers

- The **Accessibility Evaluator Resultant** that is responsible for comparing the
  pre-defined thresholds with the current values of the accessibility features of
  the SDL system, displaying and storing the results of the accessibility
  evaluation and making the suggestions to increase application’s accessibility.
  It provides input for the generation of the Accessibility Assessment Report. It
  also makes suggestions of how to increase the accessibility level of the SDL
  application. The results of the evaluation are provided as input for the
  generation of the Accessibility Assessment Report, which can be presented to
  users. The result is produced for every feature and is “Pass” if the feature is
  accessible or “Fail” if it is not. In this second case, a suggestion of how to pass
  the accessibility test is also made. To satisfy the above functionality, the
  module communicates with the Accessibility Features Parser and Accessibility
  Threshold Controller modules.

The Accessibility Assessment report informs the user for an SDL application’s
features accessibility result and if the result contains some errors and/or warning, it
 can suggest possible modifications that can be made to the SDL application in order
to make it accessible. The evaluation results will be provided as input to the module
that is responsible for interfacing with the EARL tool in order to generate the final
Accessibility Assessment report. An example of the results text file is presented to
following figure 17

```
***********************************************************************
Evaluation Date: xx/xx/xxxx
Evaluation Time: xx:xx
Application Name: SDL Camera
Assessed Feature: Window Size
Accessibility Result: Fail
Suggestions
Sug: Enlarge the window length. The window must be at least length = 1024 and
width = 860
***********************************************************************
```

**Figure 17. Example of the assessment report text file**

### 3.2.2 ACCESSIBLE ontological Knowledge Resource

The ACCESSIBLE ontological Knowledge Resource acts as a registry of knowledge
for all the other components of the ACCESSIBLE project. All ACCESSIBLE
components access the knowledge base by interacting with the ontological knowledge
resource through available interfaces.

The main goal of the ACCESSIBLE ontological knowledge is to provide support for
the formal and unambiguous definitions of accessibility domains, as well as the
possible semantic interactions between them. We have specified OWL ontologies to
be integrated into accessibility verification environments. This will establish a
common vocabulary for exchanging and describing the complex information that is
related to accessibility assessment of software applications (e.g. Web, Mobile, Web
services, description languages). The knowledge resource aims to formalize conceptual information about:

- The characteristics of users with disabilities, devices, applications, and other aspects that should be taken into account when describing an audience with disabilities and developing tailored software applications;
- Accessibility standards and associated checkpoints, approaches, etc.;
- Semantic verification rules to help describing requirements and constraints of audiences, and associating them to accessibility approaches and tests.

In order to cope with these goals, the architecture of the ACCESSIBLE ontological knowledge has been complied with the following requirements:

- To be as formal as possible, thus providing all the necessary definitions in a concise, unambiguous, and unified form;
- Provide information that can be easily processed by software developers and integrated into accessibility assessment processes;
- Easily adapted by software developers and other users involved in the software development process.

In the following Figure 18 we present the different slots where different ontologies are supposed to be integrated within Accessible Ontology: generic, domain-specific, and rules. Each one of these slots has specific goals within the overall architecture and, consequently, plays a different role in the context of accessibility assessment of software applications: (1) Generic Ontology provides a set of meta concepts related to accessibility that are independent from technological particularities. Such concepts include the description of users and devices, as well as more general terms to describe software applications domains; (2) Domain-specific Ontologies specify a set of instance concepts that are dependent of a particular application domain or technology; and (3) Rules Ontology provides the specific set of instance semantics that allow the translation of specific accessibility assessment procedures into properly verifiable terms that are applied to domain-specific concepts. The decoupling obtained from separating domain-specific ontologies from generic ontology provides the necessary abstraction level that allows the description of accessibility situations that are independent from technologies and, consequently, open the way to the integration of different accessibility assessment procedures that are properly tailored to particular application and technology domains.
More specifically the ontologies architecture serves as the basis for the definition of the architecture of the Rules Inference Engine (Deliverable D4.2 [31]), as presented to the following figure 19. The main components of the Rules Inference Engine module are:

1) **The Core Inference Engine**: The inference engine is responsible for binding ontological concepts to each one of the technology-specific accessibility assessment tools (Website, Mobile, WebServices, and SDL domains).
2) **The Rules User Interface**: The rules user interface lays at a higher abstraction level of both the Rules Inference Engine and the Ontological Framework, by establishing a way to specify new rules of accessibility assessment that can be created by accessibility assessors or developers, in the context of specific accessibility criteria not covered by ACCESSIBLE (e.g., company-based accessibility assessments).

3) **The Ontological framework**: This sub-component serves as the basis for the binding between disabilities or other usage scenario descriptions, and the existing set of evaluation procedures made available by the technology-specific accessibility assessment tools.

4) The technology-specific **accessibility assessment tools**: Lastly, the rules inference engine architecture copes with the actual assessment tools, in order to be able to trigger evaluations in the context of the different technologies supported in ACCESSIBLE.

As for the content flow, the Rules Inference Engine consumes and produces the following data:

**Inputs:**
- A SPARQL query which triggers the selection of a subset of concepts from the ACCESSIBLE ontology that are bound to accessibility assessment procedures (e.g., a WCAG test criterion);
- An optional application specification (e.g., a Web page, in the context of Web accessibility assessment).

**Outputs:**
- A subset of concepts related to accessibility assessment procedures, or
- The results (e.g., failure/pass) of running the tests on the supplied application specification, to be further processed by the reporting component of the ACCESSIBLE general architecture.

The development of the Rules Inference Engine will be based on the following technologies:
- JVM-based technologies and libraries;
- JVM-based meta-programming concepts;
- Groovy programming language and associated libraries;
- Jena library for ontology parsing and manipulation;
- Pellet library for SWRL-based reasoning.

### 3.2.3 ACCESSIBLE Portal/ User Interface Architecture

A more detailed view of the ACCESSIBLE general architecture that was described in previous section incorporates in an abstract yet understandable fashion the additions made to this architecture for specifying the building block of the user interface as presented in following Figure 20.
As presented in this figure the user interface of ACCESSIBLE takes two distinct instantiations according to the means of access. When access is carried out through the web the user interface portal is used while a stand alone application carries out the tasks of making the infrastructure available offline. For the user interface portal an n-tier architecture is used with a database implementation for storing application data.

On the other hand the stand alone version uses xml for storing data related to ACCESSIBLE and several plug and play components for offering facilities for project management, syntax aware editing, IDE like interaction through dockable controls etc. Each of the following sections aims at analysing each of the building blocks described above.

![Diagram](image)

**Figure 20.** Extending the ACCESSIBLE architecture with the building blocks of the user interface
3.2.3.1 ACCESSIBLE User Interface Portal & User assistant agent

In the context of ACCESSIBLE user interface portal and User assistant Agent a multi-tier (Figure 21) architecture is employed consisting of the (a) Data Access Layer, the (b) Business logic Layer and (c) the Presentation Layer. This architecture represents the core functionality of the portal to be operated in multiple setups from a single machine to multiple machines organized as web farms with the option to perform load balancing via dispatching of request to alternative machines or via the usage of cluster services for mirroring servers and therefore achieving fail safety.

![Diagram](Figure 21. ACCESSIBLE User Portal architecture)

**Data Access Layer**

For the database implementation the traditional relational model is used together with stored procedures for faster retrieval and insertion of data in the database, reducing the amount of client side processing by looking up data and maintaining key values and internal integrity. Furthermore, using stored procedures, the database server creates for each query a plan that includes all the information required to return the data effectively to the client. This plan is stored in the system’s cache, so that it can be reused when needed. Another advantage of the stored procedure is that the database server can create indexes, thus increasing the speed of interaction.

In the data access layer, the stored procedures created in the database are accessed by web services that are subsequently available to the business logic layer. The implementation of these web services incorporate an XML query descriptor mechanism that undertakes to connect to the database management system, call the appropriate predefined query using the available query description, pass the suitable parameters and return the acquired data. This module consists of a software library that communicates with the database and several xml files containing the description of the predefined queries for various system components. As illustrated in Figure 22, the XML query descriptor library module accepts the request from the upper sub-layer, reads the XML query description file, and calls the predefined query. The query
accesses the database and fetches the necessary data that are subsequently returned to the XML descriptor library module.

![Diagram: Data Access Layer]

**Figure 22:** The overall process flow in the data access layer

The XML query descriptor file is illustrated in Figure 23, and consists of the query and of parameters related information. More specifically, the XML file contains the name and the type of each query and the associated list of parameters with the necessary details (i.e., name, type, size and direction).

```xml
<ws:ComponentName>
  <Method Name="MethodName">
    <Parameter Name="ParameterName">
      <Name>ParameterName</Name>
      <Type>DataType</Type>
      <Size>DataSetSize</Size>
      <Direction ParameterDirection></Direction>
    </Parameter>
  </Method>
</ws:ComponentName>
```

**Figure 23:** The abstract structure of the XML descriptor file

Web services are responsible for providing the appropriate mechanism to transmit the data from the data access layer to the application (business) layer. The result of a web service is a SOAP (Simple Object Access Protocol) message that contains the requested information, as shown in Figure 24.
Figure 24: The result of a web-service invocation

**Business logic layer**

This layer will incorporate the functions needed to implement the application. More specifically this layer shall contain classes that form a higher level ontology specification of the database schema. The aim of this layer is to transform the data received by the web services of the data access layer to instances of the ontology specification. To this purpose, special methods are used to deserialize the data received and transform them into meaningful instances of the ontology. Additionally, this layer contains functionality that is used by the interface layer to perform certain actions. This strategy is followed in order to make the development of the higher levels of the portal easier and closely coupled with the UI functionality.

**Presentation Layer – The EAGER toolkit**

The ACCESSIBLE user interface portal employs an approach targeted to support and provide the means for the development of inclusive Web-based user interfaces (WUIs) capable to adapt to multiple and significantly different user profiles and contexts of use. This approach comes through the usage of a specialised UI library namely EAGER. The architecture employed by this software library is presented in Figure 25. The EAGER toolkit has been developed in Microsoft® Visual C# .NET and allows Microsoft® .NET developers to create interfaces that conform to W3C accessibility guidelines and which are able to adapt and interchange modalities, metaphors and user interface elements as appropriate for each individual user, according to profile information and context specific parameters. The process of employing EAGER is significantly less demanding in terms of time, experience and skills required from the developer than the typical process of developing Web interfaces for the “average” user, due to the flexibility provided for designing and implementing interfaces at an abstract task-oriented level. Using EAGER, designers
are not required to be aware of the low level details introduced in representing interaction elements, but only of the high level structural representation of a task and its appropriate decomposition into sub tasks, each of which represents a basic UI and system function. In conclusion, the EAGER toolkit offers, not only the aforementioned benefits, but also opens a more promising direction. It is clear that using a standard UI toolkit, a monolithic interface is created, whereas by using the EAGER toolkit, dynamically adaptable interfaces are generated.

The main core modules that will be offered to end users from the ACCESSIBLE portal, for providing them a complete evaluation environment consist of:

- **Accessibility Assessment modules:** This section of the portal will host all the accessibility evaluation facilities developed by ACCESSIBLE such as the web
assessment tool, the web services assessment tool, the description languages assessment tools etc.

- **Digital Library**: The digital library of resource is a User Interface module of the portal focusing on offering access to developers of interactive applications to material useful for enriching their knowledge on the specific domain. Such material contains but is not limited to collections of guidelines for evaluation, accessibility standards, single guidelines, best practice examples etc. In order to offer seamless access to a potentially vast amount of information several facilities are incorporated such as: Alternative browsing options, Resource search facility, Interaction history, Knowledge profiles, Personal collections of knowledge, Resource rating and reviewing etc.

- **Online Courses**: Online Courses is an e-learning facility used for enabling end users to create their personal courses through a collection of available material related to ACCESSIBLE such as presentations, examples, descriptions, tutorials etc. These courses can in turn be browsed through the usage of the courses navigator using a book style metaphor.

- **Message Board**: This module acts as a forum where potential end users of ACCESSIBLE can get asynchronous support by accessibility experts subscribed to the portal in the same manner with existing expertise exchange forums on the web.

- **Documents Area**: The documents area is a facility that enables the publication of files to the portal in order to facilitate the need of disseminating material to ACCESSIBLE end users such as manuals, tutorials etc.

- **Glossary**: Glossary is a module where terms, definitions, and abbreviations can be accessed. The collection of terms shall be populated with material related to ACCESSIBLE, and to the accessibility evaluation and testing of interactive applications.

- **Downloads**: The Downloads section is aimed for disseminating the results of ACCESSIBLE in terms of tools that can be installed and used in the end user’s computer such as the developer designer aid module.

### 3.2.3.2 ACCESSIBLE Stand Alone Interface

In this section the architecture used in the context of the user interface of the stand alone tool to act as a front-end of the ACCESSIBLE infrastructure is presented (figure 26) focusing on highlighting the building blocks and analysing specific implementation details and decisions taken during its development.
The General architecture consists of the project administration component which uses xml datasets and schemas to present the ACCESSIBLE project types and xml data adapters for using the data saved in xml data storage. These project types use the facilities offered by the Microsoft .Net framework for creating xml schemas and loading - saving data that represent these schemas via xml datasets. For each evaluation session a project file in xml is created containing all the required by the UI information for storing and restoring session’s information. An example of such a project file is presented in Figure 27.

```
<xml version="1.0" encoding="UTF-8">
<project>
  <title>Figure 27: An example of an ACCESSIBLE xml based project specification</title>
  <documents>
    <document>
      <title>Carmaggin - Wikipedia, the free encyclopedia.htm</title>
    </document>
    <document>
      <title>Leonardo da Vinci - Wikipedia, the free encyclopedia.htm</title>
    </document>
    <document>
      <title>Diamond - Wikipedia, the free encyclopedia.htm</title>
    </document>
    <document>
      <title>Carmaggin - Wikipedia, the free encyclopedia.htm</title>
    </document>
  </documents>
</project>
```
More specifically the presentation of the stand alone tool uses the following components:

- **Syntax aware editor library**: It is considered particularly important especially for developers which are the main targets of ACCESSIBLE to be able not only to evaluate interactive applications but also have the option to edit and correct in place specific errors and warnings that might occur. Thus, an Editor with syntax highlighting is being proposed in the UI in order to offer the option of editing the evaluated documents supporting the main formats currently supported by ACCESSIBLE such as HTML, CSS, WSDL, XML, SDL etc.

- **Dockable containers library**: As was already highlighted by the design of the stand alone tool the provision of a look a feel that simulates the facilities offered by IDEs was considered important especially for developers who are used in using such applications. To this end a UI library that allows the creation and usage of dockable containers can be used allowing the existence of two types of UI widgets in each interface instantiation: (a) Toolbox styles widgets that can be docked on either side of the application (pined, released, browsed as tabs etc), (b) Document style widgets that are loaded in the middle of the application and can be accessed in several ways (split horizontal views, split vertical views etc)

- **Internet explorer and/or Firefox integration component**: Using the same rational as the one that lead to the incorporation of an editor with syntax highlighting the ability of previewing evaluated documents via the usage of their rendering applications were considered important. To this end specific controls will be used in order to wrap the functionality offered by several web browsers such as internet explorer and firefox and therefore offer the ability to developers to user the user interface of ACCESSIBLE for accessing in place the evaluated documents.

### 3.2.4 ACCESSIBLE Developer/designer–aid module

The Accessible developer/designer-aid module is the one responsible for supporting target users (developers and/or designers) to facilitate the design and development of accessible software applications. It communicates with (a) the Accessible Ontology in order to get information regarding the implemented impairments and the various factors that affect each impairment and the (b) Application Module or the Netbeans IDE depending on whether the user is using the standalone or the Netbeans plugin version of the developer/designer-aid module. The plugin is able to handle both GUI forms during the development phase and GUI applications during execution phase.

If considered as a black box, the aid module takes as input a java swing GUI application bundled in a jar file or a java swing GUI form, the implemented impairments and the corresponding controls defined within the Accessible Ontology and produces as output the result of the Simulation process. The architecture of the developer designer aid module is depicted to the following figure 28.
As depicted in the architectural Figure 28, the Accessible Developer/designer-aid module is consisted of the following main components:

1. **The User Interface Layer**: This component is a Java-based application which is the bridge between the user and the core simulation layer. It is responsible for showing the running and simulated java swing GUI application or form as well as the various impairments and their controls.

2. **The Core Simulation Layer**: This component is a Java-based application which takes as input:
   
   a) Information derived from the running application or form  
   b) Information derived from the impairment chosen  
   c) Information derived from the impairment controls chosen

The overall simulation process which is depicted in the following Figure 29, consists of the following steps:

1. **Load the Java Swing GUI application in the JVM.**

   During this initial step, the user provides the java swing GUI application bundled in a jar file to the Application module. In the case of the Netbeans plugin this is done through the Netbeans IDE. Then a custom java class loader is used to load the application and the developer/designer-aid module in the same JVM. This is done so that the JAAPI can be used by the module. If the designer/developer wants to simulate a single java swing GUI form then this step is not needed and he uses the Netbeans IDE to input the form in the aid module.
2. Choose which impairment to simulate from the ones defined in the Accessible Ontology.

Within this step, the developer/designer uses the Impairment Chooser in order to specify the desired impairment. The set of the provided impairments are extracted from the Accessible Ontology.

3. Define the values for the factors that affect the impairment from Step 2. The factors are defined in the Accessible Ontology.

Once an impairment is chosen from Step 2 the corresponding Impairments Control panel is activated. This panel contains controls for the various factors that affect the chosen impairment and can be altered at will in order to simulate different degrees of severity for that impairment. These factors are also taken from the Accessible Ontology.

4. Produce the simulated version of the application or form based on the combined information from the previous steps.

Within this step, all information acquired from the previous steps is used in order for the Simulation Engine to produce the Simulated Application or Form. The Engine utilizes the JAAPI to have access to the running application. All events invoked by the application are caught by the JAAPI and trigger a new call to the Engine making a real-time simulation possible. If the designer/developer wants to simulate a single GUI form then the JAAPI is not used as the Engine uses the Netbeans IDE to have access to the GUI form.

### 3.2.5 The EARL-based reporting tool

The architecture of the ACCESSIBLE EARL(Evaluation and Reporting Language) -based Reporting tool is depicted in the following figure 30. This tool receives input from the ACCESSIBLE Assessment Simulation module and interacts with the User Interface (Web and standalone) of the assessment module. Apart from this, the EARL-based reporting tool is also provided from the Accessible User Interface Portal supplementary information regarding the context of the given evaluation procedure. In particular, information such as the Assertor’s details (username, current date, time etc) is passed from the user interface to the reporting tool. This information can be used in conjunction to the initial Assessment result, in order for the final EARL-based Report to be generated.

The tool’s outcome is an EARL-based report containing information as defined within the EARL specification, regarding the Assertor, the Subject, the Testcase and the Evaluation of the specific assessment procedure. This information is thereafter used from the Accessible User Interface Portal, in order for the Accessibility
Evaluation results to be presented to the users. Regarding the stand-alone version of the tool, the “Accessible User Interface Portal” block shown in Error! Reference source not found.30 will be replaced from the stand alone application. Although in this case the tool’s functionality will remain the same as described above, the supplementary information incorporated within the final EARL-based accessibility assessment report may differ. This is due to the fact that the stand alone application may have the ability to provide different types of information regarding the accessibility assessment’s context, than the ones provided from the Accessible User Interface Portal.

Figure 30 Accessible EARL-based Reporting Tool Overall Architecture
4 Conclusions and next steps

This deliverable provides an overview of the ACCESSIBLE architecture as a starting point and reference for further work. It describes the work completed until now, and the way forward. In this document the high-level architecture from the DoW was taken and combined with the requirements and user needs to develop a new, more refined architecture. In addition to this, a logical layer-based view of the architecture has been given and connected to the components, defined by the architecture.

At the moment, the first draft of the Architecture specification is completed allowing deliverable D3.3b [32] to use the results of this document as a starting point. Although the first draft of the ACCESSIBLE architecture has been defined, there are still quite a number of decisions to be taken and quite some details to be defined. This will be performed within the final version of this deliverable (D3.3 b). This D3.3.b deliverable will specify the detailed functionalities of each of the component and it will also list possible technologies that can be used to realize those functionalities. In addition, it will consider the constraints and limitations coming from the technology and it will specify communication protocols, parameters and response values of all interface methods. Also based on the Use case definitions that will be included to D2.4 at month 15, appropriate sequence (UML) diagrams will be included to the D2.3b.
5 Annex

5.1 SAFIRE

The SAFIRE tool will be used for the development of the SDL application that will be assessed for its accessibility due to the project’s requirements.

SAFIRE is a fully integrated development and run-time environment optimized for the implementation, validation and observation of event-based systems. It is used for a wide range of telecommunication applications, such as gateways, event-based testers & protocol analyzers and it is based on international standards such as SDL, MSC, ASN.1 & TTCN (ITU-T, ETSI, ANSI, ISO).

SAFIRE contains a graphical development environment for creating, editing & building event-based systems, test harnesses & test suites. The advanced testing features that SAFIRE contains cause the event-based systems to be validated to various levels of confidence, from top-level tests to detailed conformance tests according to international standards.

For the observation of event-based systems, SAFIRE has a built-in protocol analyzer. It is able to capture system internal signalling events, as well as signalling traffic from live networks, using SAFIRE compatible probes and 3rd party applications. The captured events are decoded, formatted and presented according to configurable protocol stacks.

There are SAFIRE compatible libraries, test suites, drivers and hardware available for a wide range of signalling systems, including mobile, internet, aerospace and trunk networks.

The SAFIRE tool chain has a modular architecture:

- SAFIRE Designer - graphical editor, viewer, compiler
- SAFIRE Campaigner - test execution & report generator
- SAFIRE Animator - slow motion replay (actions, events, behaviour)
- SAFIRE Tracer - protocol analyzer
- SAFIRE Organizer - version control & project management
- SAFIRE VM Virtual Machine - high performance virtual machine

The SAFIRE Designer allows event-based systems to be viewed in several different ways, each one always up-to-date and automatically generated. The key features of the SAFIRE Designer are:

- Graphical design of SDL state machines & test suites
- Intuitive, colorful, auto-prompting, auto-layout
- System architecture; instances, connections, interfaces
- Flow charts; inputs, outputs, actions, decisions, timers, state transitions
- State-transition diagrams; states & possible paths between them
- State-input matrix; states & coverage of possible input events
• Arrow diagrams; input events, timers & outputs

Figure 31: SAFIRE Editor sample

The designer allows SDL event-based systems to be viewed in several different ways.

• The system architecture shows the state machine instances, their nesting, the connections between them and their interfaces.

• The behavior of individual state machines is presented clearly as a flow chart including input events, outputs, actions, decisions, timers & resulting state transitions.

• The state-transition diagrams show the different states & all possible paths between the states, i.e. driven by input events.

• The state-input matrix presents all the states & the corresponding coverage of possible input events, i.e. highlighting which events are not handled.

• The arrow diagrams are an alternative view of the behavior showing input events, timers & outputs as arrows between the different FSMs.
The SAFIRE tool is not capable only for designing SDL systems but also it is able to execute them. The execution is responsibility of the SAFIRE Virtual Machine. The key features of the SAFIRE VM are:

- High performance virtual machine for SDL event-based systems.
- Direct execution, no code-generation needed
- Performance/memory statistics & diagnostics

5.2 ASK-IT Alignment tool

The purpose of this service alignment tool is to enable Web Service providers to integrate their services with the ASK-IT system (figure 33). Within the ASK-IT project’s context, the alignment of the service refers to the matching of the service’s concepts to the ASK-IT ontology’s ones, in a way that the system can use the service’s functionality and offer it to the users. The alignment tool allows software developers to align the functionality of their products, following its conceptual definition, with the aid of ontologies. With the use of the tool, a service provider that wants to integrate a service to ASK-IT first of all can view the description of these models. Then, he/she can find the model that is more appropriate to his/hers service and proceed to the “alignment” part of the integration.
In order to start the alignment process, the tool must first of all be aware of the service’s concepts. This is done by the parsing of the service’s WSDL (Web Services Description Language) file, which is given from the provider to the tool. Once the system is aware of the service’s concepts, it gives the provider the ability to align them to the ones defined in the ASK-IT services models. The alignment of the concepts is made with the use of a web-based drag-and-drop functionality (Figure 34). The mappings that come up after the alignment are stored and are used for the integration of the service to the ASK-IT Data Management Module.

Figure 33 CERTH-ITI’s ASK-IT Service Alignment Tool’s service models

Figure 34 Alignment of a service
5.3 SET Tool

The Structure Evaluation Tool (SET tool) is a software tool to analyze the validation of the structure of Web sites, in terms of accessibility structure and quality of HTML applications (following figure 35). In addition SET can visualize failure detection capabilities, based on coloured highlighting (such as sequence of headlines, missing attributes in titles or graphics, control of table layouts, language break, etc.). The SET tool can only provide to developers or designers appropriate reports (by displaying appropriate failures) regarding accessibility and quality failures.

The SET has been implemented by using Cascading Style Sheets (CSS). Thus the CSS file of this tool can collaborate with adequate toolbars included in Web browsers (e.g. the developer toolbar for the firefox browser). In this way it is possible to detect possible failures, (e.g. text which is not included in paragraph tags).

![Figure 35. SET tool](image-url)
References

[1] D2.2b: User Needs and System Requirements (final version)
[5] D2.4: Use cases
[16] http://www.w3.org/RDF/
[18] http://www.w3.org/TR/owl-features/
[22] http://www.w3.org/TR/wsdl
[26] http://www.w3.org/WAI/ER/
[27] http://validator.w3.org/
[28] http://www.w3.org/TR/CSS2/
[29] http://www.w3.org/Style/CSS/SAC/
[31] D4.2: A software package containing a set of modelling tools, rules inference engine, and the rules graphical editor

[32] D3.3b: ACCESSIBLE system architecture specification (final)