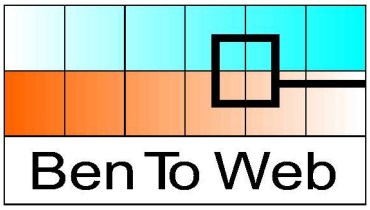
Benchmarking Tools and Methods for the Web (FP6—004275)



Sixth Framework Programme
Information Society Technologies Priority

D5.1 Preliminary open plug-in interface specification and EARL assessment

Contractual Date of Delivery to the EC:	28th February 2005
Actual Date of Delivery to the EC:	31st March 2005
Editor(s):	Carlos A Velasco, Johannes Koch (FIT)
Contributor(s):	Carlos A Velasco, Johannes Koch, Yehya Mohamad, Dirk Stegemann (FIT), Christophe Strobbe (KULRD)
Workpackage:	5
Security:	Public
Nature:	Report
Version:	C
Total number of pages:	32

Keywords: Web accessibility, evaluation and report language, EARL, Web portal, Resource Description Framework, RDF, imergo, Web Service interface.

Table of Contents

List of Figures

1 Executive Summary

This document presents an overview of the status of the Evaluation and Report Language (EARL; Chisholm and Palmer, 2002) and an analysis of its current limitations and open issues. The document is organised as follows. Section 2 gives a short introduction to EARL and a set of probable use cases that can be used to infer some additional requirements. It also includes an overview of existing RDF parsers. Section 3 describes a set of open issues that need to be tackled when bringing the specification to an usable status, as well as ongoing discussions on its structure and the processing model to be used. Finally, section 4 extracts some conclusions and outlines future work. In Section 5, we list key references for this document.

The deliverable contains four additional appendices. In Section 6, we introduce briefly RDF and point to further references. RDF is the framework in which EARL and other semantic Web applications are based. Section 7 presents an technical introduction to the actual Working Draft of EARL. Section 8 includes the actual EARL RDF schema. Finally, Section 9 presents a technical overview of the Web Services interface that will be used to make available for testing the new BenToWeb modules to the observatory of the EIAO project. The selection of this type of interface instead of a traditional API, is due to its flexibility and easiness of implementation. Once this interface has been thoroughly tested, it will be offered for public comment.

2 Introduction

The Evaluation and Report Language (EARL[1]) is a framework to express test results. The concept of test under EARL is taken in its wider acception, and can include bug reports, software unit tests, test suite evaluations, and conformance claims, among others. EARL is based upon RDF (see Section 6 for further references).

The test subject is not restricted within EARL either, and it might be basically anything "identifiable" by a URI (whether existing or not). Typical examples are Web resources, authoring tools, user agents, desktop applications, etc. The third cornerstone of EARL is the tester, which can also be a physical person or a tool.

The objectives of EARL are to:

- Create a standardised way to produce test reports;
- Support the exchange of reports between testers (humans or testing tools);
- Facilitate the comparison of test results; and
- Ease the aggregation of test results (e.g., like a different set of tests on the same subject).

As it can be seen, EARL is not only targeted to Web accessibility evaluations. Its scope is much wider, and accessibility just happens to be a usage scenario. It is also obvious that EARL is mainly targeted to the exchange of test results in a meaningful way between software applications, although it also facilitates the work of human testers.

2.1 EARL and the testing process

A generic testing process has many typical steps. Figure 1 represents a typical end-to-end process that goes from the requirements capture phase up to the processing of the test results. The early parts of the graph are connected to the elaboration of test suites and other generic approaches (see WP4 for further work in this area), whilst EARL fits clearly on the reporting of test results.

EARL is flexible enough to respond to the needs of a variety of audiences involved in a testing or quality assurance process. Typical profiles are:

- Product manager: responsible for delivering a given product;

[1] http://www.w3.org/TR/EARL10/

- Product designer: designs the product and documents this in a design specification;
- Quality engineer or tester: takes the product through a series of tests to find bugs; and
- Developer: creates a product to satisfy the design specification; fixes bugs found by the quality engineer or tester.

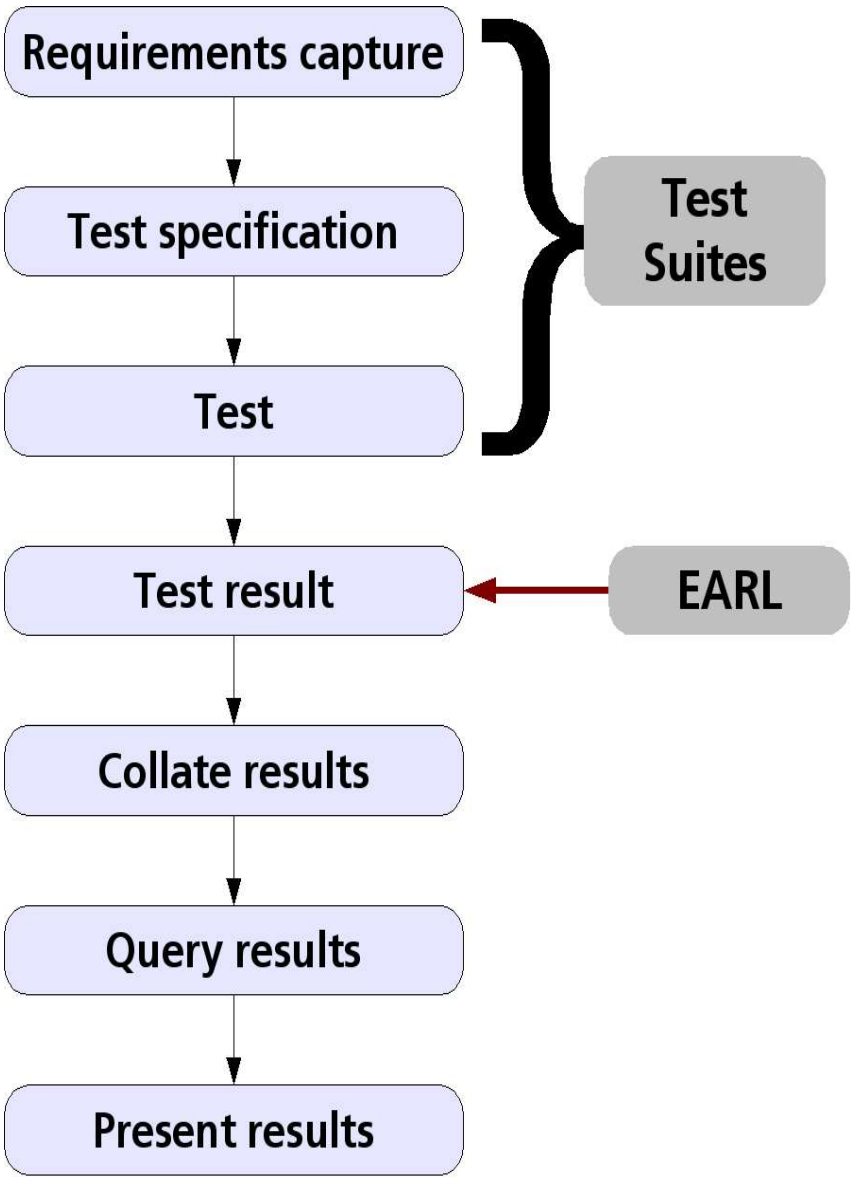


Figure 1. Graphical representation of a typical testing process.

Therefore, EARL can be seen as a powerful tool to support any quality assurance process.

2.2 Use cases

To create a solid base for EARL, bearing in mind its ambitious objectives, a set of use cases have been drafted from which a set of requirements can be inferred. Typical scenarios are:

2.2.1 Evaluating a Web site using tools in different languages

A group of people speaking different languages are evaluating a Web site for conformance to different legal environments, such as, e.g., Section 508[2] in the USA and BITV[3] in Germany. The use of EARL:

- allows for detailed messages in different languages explaining where problems are met. The report can contain messages in the languages spoken by the evaluators so that each of them understands the messages.
- allows for "keywords" to express the conformance level reached by the Web site that are language-independent. Thus a software tool can translate the validity levels in different languages.

2.2.2 Combining results from different evaluation tools

A Web site evaluator uses different tools for the task. Each tool can perform specific tests that the other tools cannot do. The evaluator's client wants a complete evaluation report. All the evaluation tools used produce a report in EARL format. Therefore, the evaluator can combine the separate reports into one bigger report, query the results, and offer to her customer statistical reports and a detailed conformance claim that specifies where the Web site does not meet the required level.

2.2.3 Comparing results from different tools against each other

A Web site evaluator uses different tools for evaluation. The tools perform the same tests. All the evaluation tools used produce a report in EARL format. Therefore, the evaluator can compare the results from different tools to increase the confidence level of the test results. It will also help to make assertions about a given resource, when one of the tools is only able

[2] http://section508.gov/
[3] Barrierefreie Informationstechnik-Verordnung (BITV): http://bundesrecht.juris.de/bundesrecht/bitv/index.html

to give a warning on a problem, but the other performs a thorough test that removes the aforementioned warning.

2.2.4 Comparing results from an evaluation tool against a test suite

For a benchmarking test of a given provider, different tools perform their tests on sample documents from a test suite. Some evaluation tools may produce false positives or false negatives. All of them create an EARL report with the result. Comparing the results of the tools with a theoretical output file from the test suite, evaluation tools could be rated according to accuracy against the test suite.

2.2.5 Monitoring a Web site over time (quality assurance)

A Web project manager wants to track the accessibility of a Web site over time by comparing current test results with previous ones. The reports contain the date/time of the tests and a way to locate the parts of the document the messages refer to. By comparing messages referring to the same locations the project manager can monitor possible improvements, and allocate resources to solve problems in the critical areas of the Web site.

2.2.6 Exchanging data with repair tools

A repair tool (or a similar module in an authoring tool) uses the results of an evaluation tool to identify the parts of the document that need to be fixed. For each instance of an error it provides a way for the user to notice the error and fix the document.

The same scenario can be used for Content Management Systems that wish to integrate an evaluation tool into their workflow, helping to locate accessibility and validation problems to Web editors.

2.2.7 Exchanging data with search engines

A search engine uses a third-party service which publishes EARL reports of Web sites.

- The user interface lets the user choose between different levels of accessibility. The list of search results contains only documents with a chosen accessibility level.

- The search engine uses the test results in the calculation of the ranking/relevance, so that it affects the search results order.

2.2.8 Extending EARL statements

A tool developer wants to have more information in the report than is defined for standard EARL. Because she still wants to be compatible with existing EARL consuming tools, she subclasses the EARL result types to provide more granularity within the tool.

2.2.9 Summary

The list of cases presented here is not meant to be exhaustive, and it is mostly focused on Web accessibility, however, it gives an overview of the needs of the potential users of EARL. Other usage scenarios can be found in http://www.w3.org/TR/EARL10/#user-scenarios.

2.3 EARL implementations and available RDF tools

In the document "Evaluation, Repair, and Transformation Tools for Web Content Accessibility,"[4] the reader can find an exhaustive list of accessibility-related tools. The document details which of them are able to produce EARL output. It must be highlighted that although several tools are currently able to produce EARL reports, most of them are "interpreting" EARL due to the looseness of the specification at the moment.

Since to the knowledge of the authors, no EARL parsers are available, we want to mention here a set of existing RDF parsers, which could be used to build such a tool. The selection has been made on the basis of the platform in which they run and their popularity.

2.3.1 Jena

The Jena Java RDF API[5] is a comprehensive Semantic Web framework for Java developed by Brian McBride, Jeremy Carroll, Andy Seaborne, Dave Reynolds and Ian Dickinson, from HP Labs (Bristol) with other contributors from all over the world. Jena contains an API for manipulating RDF models, including statement and resource-centric methods using

[4] http://www.w3.org/WAI/ER/existingtools
[5] http://jena.sourceforge.net/

cascading calls for easy object orientated use. It is available under an open source, BSD-like license.

Jena provides a programmatic environment for RDF, RDFS[6] and OWL,[7] including a rule-based inference engine. The Jena Framework includes:

- A RDF API
- Reading and writing RDF in RDF/XML, N3 and N-Triples
- An OWL API
- In-memory and persistent storage
- RDQL – a query language for RDF

2.3.2 Redland RDF Application Framework

Redland[8] is a set of free software packages that provide support for the Resource Description Framework. Among its features, we can highlight:

- Modular, object based libraries written in C.
- APIs for manipulating the RDF graph, triples, URIs and Literals.
- Triple sequences for efficient streaming.
- Parsers and Serializers for reading and writing RDF as RDF/XML, N-Triples and Turtle Terse RDF Triple Language syntaxes.
- Storage for graphs in memory, or with persistence mechanisms.
- Querying with RDQL and SPARQL using the Rasqal RDF Query Library.
- Redland contexts for managing data aggregation and recording provenance.
- Language Bindings in C#, Java, Obj-C, Perl, PHP, Python, Ruby and Tcl via the Redland Bindings package.
- Command line utility programs.
- Portable, fast and with no known memory leaks.

All Redland packages are free software/open source software and released under the LGPL 2.1, GPL 2 or Apache 2 licenses as alternatives.

[6] http://www.w3.org/TR/rdf-schema/
[7] Web Ontology Language: http://www.w3.org/2004/OWL/#specs
[8] http://librdf.org/

2.3.3 Drive

Drive[9] is an RDF parser written in C# for the .NET platform. It is fully compliant with the W3C RDF syntax specification and is available as open source under the terms of the GNU LGPL license.

[9] http://www.driverdf.org/

3 EARL Evaluation: existing limitations

At the timing of writing this deliverable, there is a frantic activity within the ERT WG to advance the language to a usable status. In this section we will review some of its limitations and known issues.

3.1 Current schema issues

Several people have noted that the RDF schema of the Working Draft does not validate. This issue can be easily solved (see Section 8). Additionally, the specification does not delve into too many details in regard to the properties of some classes, and how to handle different situations (see following sub-sections).

3.2 Subject of a test

With the current specification, it is unclear how to define the subject of a test. The specification leaves undefined whether the subject of a test is a given Web resource (or tool, or user agent), to which the test is applied, or whether it is the fragment of the document (or a given functionality) in which really the test has some effect.

For example, let us define a scenario in which we are a testing an HTML document with two images. The test consists in determining whether these images have an `alt` attribute, and the result is that it is not found on any them. From a quick analysis, there are three possibilities to express this within the actual specification:

- Version 1:
  One Assertion, one subject (URL of document), one testCase (alt attribute test), one result (fail) with two messages (about each of the images, including location information).
- Version 2:
  Two Assertions, each with same subject (URL of document), same testCase, one message each (including location information).
- Version 3:
  Two Assertions, different subject (URL of document + xpointer of error instance), same testCase, one message each.

The situation could become more complex in case that, for instance, you need to apply the same test to a collection of Web resources (e.g., if you are checking navigation consistence between all of them). Then, some kind of extension mechanism will need to be foreseen.

3.3 Result location

The location of test results is even a more challenging task. Given the wide variety of resources in the Internet (plain text, markup documents -not necessarily XML-, binary resources, and collections thereof), it is not possible to define a single location mechanism.

There are many alternatives, but probably the most feasible approach will be to allow for different location approaches according to the MIME type[10] of the document. Considered approaches are:

- Text documents (e.g., source files). Document fragments could be identified via line and column numbers.
- Markup documents (both HTML and XML). Several mechanisms could be used:
  - As in the previous case, both line and column numbers could be used, although that will not be a very useful information when processing the document automatically (e.g., via DOM).
  - XPath or XPointer expressions could be used, although again accuracy problems will be met for HTML documents when creating their DOM representation, or for ill-formed XML documents. An attempt could be to use "fuzzy pointers.[11]"
- Binary documents. A customised approach needs to be defined.

The issue of location is critical for a reliable comparison of test results. Therefore, the WG will need to examine this in detail.

3.4 Test identification

To be able to compare and benchmark different tools and resources, the test applied will need to be uniquely identified. In the current specification, this is a property of the Assertion class. However, it might be necessary to introduce more flexibility into this concept (e.g., a test case composed of several other sub-cases), and probably define a class for it.

[10] http://www.iana.org/assignments/media-types/
[11] http://lists.w3.org/Archives/Public/w3c-wai-er-ig/2002Feb/0002.html and http://lists.w3.org/Archives/Public/w3c-wai-er-ig/2002Feb/0009.html

3.5 Reusing components from other vocabularies

An additional open issue from the current EARL specification is to deprecate metadata elements that are defined in other well-known specifications, like for instance:

- Dublin Core Metadata Initiative:[12]
  The Dublin Core is a metadata standard for describing digital resources, often expressed in XML. The first standard published is the Dublin Core Metadata Element Set.[13] It consists of 16 optional metadata elements, any of which may be repeated or omitted. Typical elements are: Title, Creator, Subject, Description, Publisher, Contributor, Date, Type, Format, Identifier, Source, Language, etc. The Dublin Core Metadata Element Set was accepted as a NISO standard in 2001 (ANSI/NISO Z39.85-2001)[14] and as an ISO standard in 2003 (ISO 15836:2003(E)).[15]
- Friend of a Friend (FOAF) project:[16]
  The FOAF project is about creating a Web of machine-readable resources describing people, the links between them and the things they create and do. Of particular interest for EARL are the Classes foaf:Person and foaf:Project (see specification at: http://xmlns.com/foaf/0.1/).

3.6 EARL structure

A proposed structure to extend the actual set of EARL classes could be:[17]

- Subject (TestSubject)
  See previous considerations (section 3.2) on open issues.
- Location
  See previous considerations (section 3.3) on open issues. There is a clear need for this element, and thus far, different vendors and developers have opted for different approaches.
- Test Case
  See previous considerations (section 3.4) on open issues.

[12] http://dublincore.org/
[13] http://dublincore.org/documents/dces/
[14] http://www.niso.org/standards/resources/Z39-85.pdf
[15] http://www.niso.org/international/SC4/n515.pdf
[16] http://www.foaf-project.org/
[17] Proposed by the Chair (http://lists.w3.org/Archives/Public/public-wai-ert/2005Apr/0001.html). Not endorsed yet by the WG.

- Evidence
  Proposed as a mechanism to describe the sub-tests carried out for a Test Case.
- Confidence
  Proposed to describe how the different test cases were carried out.
- Message
  Information for human consumption that shall include localisation to different languages.

4 Conclusions

This deliverable has reviewed the status of EARL, as of its current Working Draft, and outlined a set of open issues that are being tackled by the WG. BenToWeb representatives are involved in this process, and will incorporate the outcomes of the WG in both specific BenToWeb tasks (like those of WPs 4 and 5), and relevant WAB-Cluster work related to the unified methodology.

5 References

Berners-Lee T, Hendler J, Lassila O (eds) (2001). The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American, May 2001. Available at: http://www.sciam.com/print_version.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21

Caldwell B, Chisholm W, Vanderheiden G, White J (2004). Web Content Accessibility Guidelines 2.0, W3C Working Draft 19 November 2004. World Wide Web Consortium (W3C). Available at: http://www.w3.org/TR/WCAG20/

Chinnici R, Gudgin M, Moreau J-J, Schlimmer J, Weerawarana S (eds) (2004). Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Working Draft 3 August 2004. World Wide Web Consortium. Available at: http://www.w3.org/TR/wsdl20

Chisholm W, Vanderheiden G, Jacobs I (eds) (1999). Web Content Accessibility Guidelines 1.0, W3C Recommendation 5-May-1999. World Wide Web Consortium (W3C). Available at: http://www.w3.org/TR/WCAG10/

Chisholm W, Palmer S B (eds) (2002). Evaluation and Report Language (EARL) 1.0. W3C Working Draft 06 December 2002. World Wide Web Consortium. Available at: http://www.w3.org/TR/EARL10/

Manola F, Miller E (eds) (2004). RDF Primer, W3C Recommendation 10 February 2004. World Wide Web Consortium. Available at: http://www.w3.org/TR/rdf-primer/

McCathieNevile C (2005). RDF for XMLers. In: Delgado-Kloos C, McCathieNevile C (eds), XML Today. UPGRADE Vol. VI, No. 1, February 2005, pp. 39—44.

Mitra N (ed) (2003). SOAP Version 1.2 Part 0: Primer. W3C Recommendation 24 June 2003. World Wide Web Consortium. Available at: http://www.w3.org/TR/soap12-part0/

RFC 1945: Hypertext Transfer Protocol – HTTP/1.0. Available at: http://www.ietf.org/rfc/rfc1945.txt

RFC 2068: Hypertext Transfer Protocol – HTTP/1.1. Available at: http://www.ietf.org/rfc/rfc2068.txt

6 Appendix: Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a language based on XML for representing information about resources in the World Wide Web (Manola and Miller, 2004). We will not provide within this document detailed information about RDF, but a brief overview to give some context to this deliverable, and EARL in particular. For further information, please refer to Manola and Miller (2004), and pointers therein, and to McCathieNevile (2005).

RDF is particularly intended for representing metadata about Web resources, and as such is the basis for the Semantic Web. By generalizing the concept of a "Web resource," RDF can also be used to represent information about things that can be identified on the Web (by means of a virtual URI), even when they cannot be directly retrieved on the Web.

The key objective of RDF is to provide "machine-readable" description of objects (although it can also provide descriptions targeted to humans). RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools.

RDF is based on the idea of identifying things using Web identifiers (called Uniform Resource Identifiers,[18] or URIs), and describing resources in terms of simple properties and property values. As it can be seen in the following listing (after McCathieNevile, 2005), RDF is basically a model for describing information in statements which are expressed in three parts, elements or triples,[19] with the form:

- something (called the subject or resource);
- has a specific relationship (called the predicate or property) with;
- with something else (called the object or value of the property).

```
<rdf:RDF
  xmlns:rdf= http://www.w3.org/1999/02/22-rdf-syntax-ns#
  xmlns:dc= http://purl.org/dc/elements/1.1/
  xmlns:foaf= http://xmlns.com/foaf/0.1/ >

  <rdf:Description rdf:about= http://www.bartleby.com/173'>
    <dc:creator>Albert Einstein</dc:creator>
    <dc:language>en</dc:language>
    <dc:title>Relativity The Special and General Theory</dc:title>
```

[18] http://www.ietf.org/rfc/rfc2396.txt
[19] Also represented as a graph of nodes and arcs.

```
    <dc:date>1920</dc:date>
    <dc:identifier>http://www.bartleby.com/173</dc:identifier>
    <dc:creator>
      <foaf:Person>
        <foaf:name>Albert Einstein</foaf:name>
        <foaf:birthday>3 June 1880</foaf:birthday>
      </foaf:Person>
    </dc:creator>
  </rdf:Description>

</rdf:RDF>
```

RDF is the basis, together with a whole set of recommendations,[20] of the Semantic Web (Berners-Lee et al., 2001). The Semantic Web is a step forward in the Web as we know it, totally targeted to human consumption. Nowadays, computers can adeptly parse Web pages for layout and routine processing, but in general, computers have no reliable way to process the semantics of the content of the page.

The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users. The Semantic Web shall not be considered a separate Web, but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation.

[20] http://www.w3.org/2001/sw/

7 Appendix: an EARL Primer

7.1 Introduction

In this appendix, we review briefly the current status of the specification as a Working Draft dated in 2002 (Chisholm and Palmer, 2002). As seen in the body of this document, EARL is far from complete, but the Evaluation and Repair Working Group (ERT WG) of the World Wide Web Consortium (W3C) is working actively to convert it into a W3C Note or Recommendation. For an in-depth review of the activities of the WG, please refer to the WG home page[21] or to deliverable D4.0/D5.0.

The aim of the Evaluation And Report Language (EARL) was to become "a general-purpose language for expressing test results." The current version of the EARL specification can be found at http://www.w3.org/TR/EARL10/. EARL is based on the Resource Description Framework (see Section 6). Understanding at least the basics of RDF is necessary for understanding and using EARL.

EARL is a format for test results, no matter who ran the tests, what subject was tested, whether the test was manual or automatic, or which sort of test was performed. Therefore, EARL could be used for expressing results of e.g., markup validity testing, accessibility testing, or testing elements of a company's corporate design.

RDF is only a model, not a language format. It can be represented in various ways like, e.g., RDF/XML[22] or Notation3.[23] The current EARL working draft does not specify which representation to use. Nevertheless, it is quite safe to assume that applications producing or consuming EARL accept the RDF/XML representation, which will be used in the following.

7.2 The EARL format

As EARL is based on RDF, an EARL report starts with the RDF element from the rdf namespace[24] and defines its own namespace.[25]

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:earl="http://www.w3.org/WAI/ER/EARL/nmg-strawman#">
```

[21] http://www.w3.org/WAI/ER/
[22] http://www.w3.org/TR/rdf-syntax-grammar/
[23] http://www.w3.org/DesignIssues/Notation3.html
[24] http://www.w3.org/1999/02/22-rdf-syntax-ns#
[25] http://www.w3.org/WAI/ER/EARL/nmg-strawman#

```
  <!-- ... -->
</rdf:RDF>
```

Then, it contains several elements (or classes). In the following, we will review each class with its properties.

7.2.1 Assertor Class

The assertor states the result of a test. An assertor can be a person or a tool. In the RDF/XML representation for EARL you use the `Person` or `Tool` elements from the EARL namespace. The Assertor class can contain the following properties (elements in RDF/XML):

- `contactInfo` (text with contacting information),
- `email` (an email address),
- `name` (the name of the person or tool),
- `platform` (the platform the tool runs on).

For example:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:earl="http://www.w3.org/WAI/ER/EARL/nmg-strawman#">

  <earl:Tool rdf:about="http://example.org/#assertor1">
    <earl:name>MyTestTool</earl:name>
    <earl:platform>FreeBSD</earl:platform>
    <earl:contactInfo>MyCompany,
      MyStreet 123,
      54321 MyTown</earl:contactInfo>
    <earl:email>mytesttool@example.org</earl:email>
  </earl:Tool>

</rdf:RDF>
```

7.2.2 TestSubject Class

Another class of elements is the TestSubject. The current working draft defines three sub-classes (elements in RDF/XML): `Tool` (a piece of software), `UserAgent` (software used to access information on the World Wide Web), `WebContent` (information on the World Wide Web). Possible properties are:

- `format` (for WebContent)
- `reprOf` (for WebContent)

For example:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:earl="http://www.w3.org/WAI/ER/EARL/nmg-strawman#">

  <earl:Tool rdf:about="http://example.org/#assertor1">
    <earl:name>MyTestTool</earl:name>
    <earl:platform>FreeBSD</earl:platform>
    <earl:contactInfo>MyCompany,
      MyStreet 123,
      54321 MyTown</earl:contactInfo>
    <earl:email>mytesttool@example.org</earl:email>
  </earl:Tool>
  <earl:WebContent rdf:about="http://example.org/#testSubject1">
    <earl:reprOf rdf:resource="http://www.google.org"/>
  </earl:WebContent>

</rdf:RDF>
```

7.2.3 TestResult Class

TestResult is a container for the result of a test and can contain the following properties:

- `validity` (a category of test result; possible values are cannotTell, fail, notApplicable, notTested, pass),
- `confidence` (a level of confidence for the stated result; possible values are low, medium, high),
- `message` (a textual message).

7.2.4 Assertion Class

Assertion has no sub-classes defined. Therefore, the element to use in RDF/XML is `Assertion`. An assertion is a statement about the results of performing a test. It can have the following properties:

- `assertedBy` (empty, with an RDF reference to an element of class Assertor),
- `subject` (with an RDF reference to an element of class TestSubject),
- `testCase` (with an RDF reference to a test case),
- `result` (containing the properties of the TestResult class),
- `mode` (with an RDF reference indicating the test mode; possible values are manual, heuristic, automatic).

For example:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:earl="http://www.w3.org/WAI/ER/EARL/nmg-strawman#">
```

```
  <earl:Tool rdf:about="http://example.org/#assertor1">
    <earl:name>Markup Validator</earl:name>
    <earl:platform>Unix</earl:platform>
    <earl:contactInfo>The W3C Validator Team,
      World Wide Web Consortium</earl:contactInfo>
    <earl:email>www-validator@w3.org</earl:email>
  </earl:Tool>

  <earl:WebContent rdf:about="http://example.org/#testSubject1">
    <earl:reprOf rdf:resource="http://www.google.org"/>
  </earl:WebContent>

  <earl:Assertion rdf:about="http://example.org/#Assertion1">
    <earl:subject rdf:resource="http://example.org/#testSubject1"/>
    <earl:assertedBy rdf:resource="http://example.org/#assertor1"/>
    <earl:testCase
      rdf:resource="http://www.w3.org/TR/html4/strict.dtd"/>
    <earl:mode
      rdf:resource="http://www.w3.org/WAI/ER/EARL/nmg-strawman#"/>
    <earl:result>
      <earl:validity
      rdf:resource="http://www.w3.org/WAI/ER/EARL/nmg-strawman#fail"/>
      <earl:confidence
      rdf:resource="http://www.w3.org/WAI/ER/EARL/nmg-strawman#high"/>
      <earl:message>
        The web content is not valid HTML 4.01 Strict.
      </earl:message>
    </earl:result>
  </earl:Assertion>

</rdf:RDF>
```

This is a quick overview of the actual EARL components. A detailed tutorial cannot be built because of the high number of existing open issues.

8 Appendix: EARL RDF schemas

The following is the EARL 1.0 RDF schema as it appears in the Working Draft (fixing a missing trailing slash that breaks validity):

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!DOCTYPE rdf:RDF [

    <!ENTITY earl 'http://www.w3.org/WAI/ER/EARL/nmg-strawman#'>
    <!ENTITY rdf  'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
    <!ENTITY rdfs 'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#'>
]>

<rdf:RDF xmlns:earl="&earl;"
        xmlns:rdf="&rdf;"
        xmlns:rdfs="&rdfs;">

    <!-- Classes -->
    <rdfs:Class rdf:about="&earl;Assertion" rdfs:label="Assertion">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&earl;Assertor" rdfs:label="Assertor">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&earl;ConfidenceLevel"
rdfs:label="ConfidenceLevel">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&earl;Platform" rdfs:label="Platform">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&earl;TestCase" rdfs:label="TestCase">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&earl;TestMode" rdfs:label="TestMode">
        <rdfs:subClassOf rdf:resource="&rdf;Resource"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&earl;TestResult" rdfs:label="TestResult">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&earl;TestSubject"
rdfs:label="TestSubject">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&earl;Tool" rdfs:label="Tool">
        <rdfs:subClassOf rdf:resource="&earl;TestSubject"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&earl;UserAgent" rdfs:label="UserAgent">
        <rdfs:subClassOf rdf:resource="&earl;TestSubject"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&earl;ValidityLevel"
rdfs:label="ValidityLevel">
        <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
    </rdfs:Class>
    <rdfs:Class rdf:about="&earl;WebContent" rdfs:label="WebContent">
```

```
        <rdfs:subClassOf rdf:resource="&earl;TestSubject"/>
    </rdfs:Class>

    <!-- Properties -->
    <rdf:Property rdf:about="&earl;assertedBy"
rdfs:label="assertedBy">
        <rdfs:domain rdf:resource="&earl;Assertion"/>
        <rdfs:range rdf:resource="&earl;Assertor"/>
    </rdf:Property>
    <rdf:Property rdf:about="&earl;confidence"
rdfs:label="confidence">
        <rdfs:range rdf:resource="&earl;ConfidenceLevel"/>
        <rdfs:domain rdf:resource="&earl;TestResult"/>
    </rdf:Property>
    <rdf:Property rdf:about="&earl;contactInfo"
rdfs:label="contactInfo">
        <rdfs:range rdf:resource="&rdfs;Resource"/>
        <rdfs:domain rdf:resource="&earl;Assertor"/>
    </rdf:Property>
    <rdf:Property rdf:about="&earl;email" rdfs:label="email">
        <rdfs:range rdf:resource="&rdfs;Literal"/>
        <rdfs:domain rdf:resource="&earl;Assertor"/>
        <rdfs:subPropertyOf rdf:resource="&earl;contactInfo"/>
    </rdf:Property>
    <rdf:Property rdf:about="&earl;format" rdfs:label="format">
        <rdfs:range rdf:resource="&rdfs;Literal"/>
        <rdfs:domain rdf:resource="&earl;WebContent"/>
    </rdf:Property>
    <rdf:Property rdf:about="&earl;message" rdfs:label="message">
        <rdfs:range rdf:resource="&rdfs;Literal"/>
        <rdfs:domain rdf:resource="&earl;TestResult"/>
    </rdf:Property>
    <rdf:Property rdf:about="&earl;mode" rdfs:label="mode">
        <rdfs:range rdf:resource="&earl;TestMode"/>
        <rdfs:domain rdf:resource="&earl;Assertion"/>
    </rdf:Property>
    <rdf:Property rdf:about="&earl;name" rdfs:label="name">
        <rdfs:range rdf:resource="&rdfs;Literal"/>
        <rdfs:domain rdf:resource="&earl;Assertor"/>
    </rdf:Property>
    <rdf:Property rdf:about="&earl;platform" rdfs:label="platform">
        <rdfs:range rdf:resource="&rdfs;Resource"/>
        <rdfs:domain rdf:resource="&earl;Assertor"/>
    </rdf:Property>
    <rdf:Property rdf:about="&earl;reprOf" rdfs:label="reprOf">
        <rdfs:range rdf:resource="&rdfs;Resource"/>
        <rdfs:domain rdf:resource="&earl;WebContent"/>
    </rdf:Property>
    <rdf:Property rdf:about="&earl;result" rdfs:label="result">
        <rdfs:domain rdf:resource="&earl;Assertion"/>
        <rdfs:range rdf:resource="&earl;TestResult"/>
    </rdf:Property>
    <rdf:Property rdf:about="&earl;subject" rdfs:label="subject">
        <rdfs:domain rdf:resource="&earl;Assertion"/>
        <rdfs:range rdf:resource="&earl;TestSubject"/>
    </rdf:Property>
    <rdf:Property rdf:about="&earl;testcase" rdfs:label="testcase">
        <rdfs:domain rdf:resource="&earl;Assertion"/>
        <rdfs:range rdf:resource="&earl;TestCase"/>
```

```
    </rdf:Property>
    <rdf:Property rdf:about="&earl;validity" rdfs:label="validity">
        <rdfs:domain rdf:resource="&earl;TestResult"/>
        <rdfs:range rdf:resource="&earl;ValidityLevel"/>
    </rdf:Property>

    <!-- Instances of Classes -->
    <earl:TestMode rdf:about="&earl;manual" rdfs:label="manual">
        <rdfs:comment>The test was performed by a
human.</rdfs:comment>
    </earl:TestMode>
    <earl:TestMode rdf:about="&earl;heuristic" rdfs:label="heuristic">
        <rdfs:comment>The test is derived from other test
results.</rdfs:comment>
    </earl:TestMode>
    <earl:TestMode rdf:about="&earl;automatic" rdfs:label="automatic">
        <rdfs:comment>The test was performed by a tool or
machine.</rdfs:comment>
    </earl:TestMode>
    <earl:ValidityLevel rdf:about="&earl;cannotTell"
rdfs:label="cannotTell"/>
    <earl:ValidityLevel rdf:about="&earl;fail" rdfs:label="fail"/>
    <earl:ConfidenceLevel rdf:about="&earl;high" rdfs:label="high"/>
    <earl:ConfidenceLevel rdf:about="&earl;low" rdfs:label="low"/>
    <earl:ConfidenceLevel rdf:about="&earl;medium"
rdfs:label="medium"/>
    <earl:ValidityLevel rdf:about="&earl;notApplicable"
rdfs:label="notApplicable"/>
    <earl:ValidityLevel rdf:about="&earl;notTested"
rdfs:label="notTested"/>
    <earl:ValidityLevel rdf:about="&earl;pass" rdfs:label="pass"/>

</rdf:RDF>
```

Charles McCathieNevile has offered to the WG a version of this schema with a normalised syntax and the xml:lang attribute for textual information.[26]

[26] See http://lists.w3.org/Archives/Public/public-wai-ert/2005Mar/0003.html

9 Appendix: Web Service interface to BenToWeb modules

9.1 Introduction

This appendix presents the Web Service interface that will be offered by the modules developed within BenToWeb. The selection of Web Services technology for this interface is aimed to simplify the interaction of these new modules with third party tools, like, e.g., the observatory of the EIAO project during the project lifetime.

We have selected Web Services, instead of other available remote procedure calls like XML-RPC,[27] because its flexibility. XML-RPC offers a simple, portable way to make remote procedure calls over the HTTP protocol[28] and implementations exist in Perl, Java, Python, C, C++, PHP and many other programming languages. However, XML-RPC is limited to exchanges with a limited number of parameters of six fixed types, plus structures and arrays, which are not enough to cover the needs of the project.

Web services are a new breed of Web applications with the following characteristics:

- They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web (in our case the invocation will be restricted).
- Web services perform functions that can be anything from simple requests to complicated business processes.

9.2 Implementation

9.2.1 Overview

We will base our implementation on two enabling technologies: XML and SOAP. SOAP (Simple Object Access Protocol; Mitra, 2004) uses XML messages to invoke remote methods. It consists mainly of the following components:

- An envelope structure that contains a message

[27] http://www.xmlrpc.com/spec
[28] HTTP/1.0 (RFC 1945) and HTTP/1.1 (RFC 2068). See section 5.

- Encoding rules for data types
- Rules for RPC requests and responses
- Rules for exchanging messages using an underlying protocol

The SOAP application's structure consists of two messages:

- A request invokes a method on a remote object.
- A response returns the result of running the method.

A typical SOAP request (version 1.1) looks like the following (header omitted):

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <m:getQuote xmlns:m="http://example.com/ns/">
      <m:symbol>ACOMPANY</m:symbol>
    </m:getQuote>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

And a SOAP response (version 1.1) like this snippet:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <m:Quote xmlns:m="http://example.com/ns/">
      <m:Price>4.12</m:Price>
    </m:Quote>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

9.2.2 Implementation

The implementation is based upon a Java Web Service running under Apache Axis.[29] Its description is based upon Web Services Description Language (WSDL; Chinnici et al., 2004).

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  targetNamespace="http://example.com/services/ImergoService"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://example.com/services/ImergoService"
  xmlns:intf="http://example.com/services/ImergoService"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <wsdl:message name="checkRequest">
```

[29] http://ws.apache.org/axis/

```
      <wsdl:part name="url" type="soapenc:string"/>
      <wsdl:part name="content" type="soapenc:string"/>
      <wsdl:part name="contentType" type="soapenc:string"/>
      <wsdl:part name="encoding" type="soapenc:string"/>
      <wsdl:part name="ruleSet" type="soapenc:string"/>
   </wsdl:message>

   <wsdl:message name="checkResponse">
      <wsdl:part name="checkReturn" type="soapenc:string"/>
   </wsdl:message>

   <wsdl:portType name="WebServiceImpl">
      <wsdl:operation name="check"
        parameterOrder="url content contentType encoding ruleSet">
         <wsdl:input message="impl:checkRequest" name="checkRequest"/>
         <wsdl:output message="impl:checkResponse"
           name="checkResponse"/>
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="ImergoServiceSoapBinding"
      type="impl:WebServiceImpl">
      <wsdlsoap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="check">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="checkRequest">
            <wsdlsoap:body use="encoded"
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="http://access.fit.fraunhofer.de/ns/imergo/2004
/report”>
         </wsdl:input>
         <wsdl:output name="checkResponse">
            <wsdlsoap:body use="encoded"
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="http://example.com/services/ImergoService"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="WebServiceImplService">
      <wsdl:port binding="impl:ImergoServiceSoapBinding"
        name="ImergoService">
         <wsdlsoap:address
           location="http://example.com/services/ImergoService"/>
      </wsdl:port>
   </wsdl:service>

</wsdl:definitions>
```

Basically, the service offers a single **check** method, with the following signature:

```
public String check(String url, String content, String contentType,
                    String encoding, String ruleSet)
```

where:

- url: is a string representing the URL of the resource to be checked (for identification purposes, nothing will be fetched).
- content: Content of the aforementioned URL. It must be embedded in a <![CDATA[...]]> section, or all especial characters must be escaped.
- contentType: MIME type of the given URL, e.g., "text/html".
- encoding: encoding of the given URL, e.g., "utf-8".
- ruleSet: string representing the selected rule-set. The tool will offer different rule-sets, and they will be identified by a unique code.

As return, the tool will generate an EARL (Evaluation and Report Language, Chisholm and Palmer, 2002) report with all errors in the page. A typical SOAP request with HTTP headers will look like the following snippet:

```
POST /axis/services/ImergoService HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related,
text/*
User-Agent: Axis/1.2RC2
Host: localhost:8080
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 1092

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <check
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <arg0 xsi:type="soapenc:string"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">http
://example.com/irgendwas.html</arg0>
      <arg1 xsi:type="soapenc:string"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
        &lt;!DOCTYPE html PUBLIC &quot;-//W3C//DTD HTML 4.01//EN&quot;
&quot;http://www.w3.org/TR/html4/strict.dtd&quot;&gt;&lt;html&gt;&lt;h
ead&gt;&lt;/head&gt;
        &lt;body&gt;&lt;h1&gt;Test&lt;/h1&gt;
        &lt;h3&gt;foo&lt;/h3&gt;&lt;/body&gt;&lt;/html&gt;</arg1>
      <arg2 xsi:type="soapenc:string"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">text
/html</arg2>
      <arg3 xsi:type="soapenc:string"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">utf-
8</arg3>
      <arg4 xsi:type="soapenc:string"
```

```
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">wcag
1.0_AAA</arg4>
    </check>
  </soapenv:Body>
</soapenv:Envelope>
```

And a typical SOAP response looks like the following:

```
<?xml version="1.0" encoding="UTF-8"?>

<soapenv:Envelope
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <checkResponse
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
          <checkReturn xsi:type="soapenc:string"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
              <![CDATA[ <EARL_Report> ]]>
          </checkReturn>
      </checkResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

9.2.3 Client example

A simple Java client using Axis libraries is presented here for the sake of completeness:

```
package org.bentoweb.dummy;

import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;

import javax.xml.rpc.ServiceException;

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;

public class ExampleClient_Check_1
{

    private final static String serviceURL =
        "http://example.com/services/ImergoService";

    private final static String documentURL =
        "http://www.irgendwo.com/irgendwas.html";

    private final static String testHTML =
```

```
        "<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01//EN\"
\"http://www.w3.org/TR/html4/strict.dtd\"><html><head></head><body><h1
>Test</h1><h3>foo</h3></body></html>";

    private final static String testMIME = "text/html";

    private final static String testEncoding = "utf-8";

    private final static String testRuleSets = "wcag1.0_AAA";

    public static void main(String[] args)
    {

        Object[] params = new Object[] {
            documentURL, testHTML, testMIME, testEncoding,
            testRuleSets };
        Service service = new Service();

        try
        {
            URL url = new URL(serviceURL);
            Call call = (Call) service.createCall();

            call.setTargetEndpointAddress(url);
            call.setOperationName("check");

            String report = (String) call.invoke(params);

            System.out.println (report);
        }
        catch (MalformedURLException e)
        {
            System.err.println (e.getLocalizedMessage());
        }
        catch (ServiceException e)
        {
            System.err.println (e.getLocalizedMessage());
        }
        catch (RemoteException e)
        {
            System.err.println (e.getLocalizedMessage());
        }

    }
}
```