SAmgI: Automatic Metadata Generation v2.0

Michael Meire, Erik Duval
Dept. Computer Science,
Katholieke Universiteit Leuven, Belgium
{michael.meire, erik.duval}@cs.kuleuven.be

Xavier Ochoa Chehab
Information Technology Center,
ESPOL-Escuela Superior Politécnica del Litoral
Guayaquil, Ecuador
xavier@cti.espol.edu.ec

Abstract: Relying fully on manual effort in generating metadata is not the way to go: we need automation of this process, as much as possible, without losing the "quality" of the metadata. In this paper we report on our experiences with automatic metadata generation. We briefly outline the first version of our framework for automatic metadata generation and then report on some lessons we learned. These lessons resulted in a redesign of the framework. An important aspect in this redesign is the bottom-up strategy, instead of the top-down design we used before. This means that we start with an abstract specification that is then made concrete in implementations. This should allow for interoperability between installations that do (part of) the metadata generation.

The redesigned framework is used in a case study that also extends the metadata generation part by allowing for search and retrieval of the generated metadata. We also report on an evaluation experiment that we set up for this case study, which points out that the automatically generated metadata has a similar level of quality than manually generated records present in the ARIADNE repository.

Finally we conclude with some future work.

1. Automatic Metadata Generation: the general architecture

1.1 Why we need automatic metadata generation

Relying fully on humans in generating metadata is not the way to go: humans "don't scale" and the metadata they create are not perfect. More importantly, it is hard to sustain manual creation of metadata. Therefore we need automation of the metadata generation process. This can go from a small amount of automation within a mainly human-based flow, up to full automation without needing any human input or reviewing.

1.2 First version of the framework

In [1] and [2] we reported on a framework for automatic metadata generation [3]. Our main focus was to show that automatic metadata generation is feasible. We implemented some case studies to prove this claim, and made a basic evaluation of the obtained results.

A second step was then to see what lessons could be learned from that first version and to develop a second version of the framework that allows easier use by people using other platforms or systems. Because integrating existing efforts and implementations is a key aspect for us, we need a solid base infrastructure, including an API on top of that, allowing other people to add functionality. Once that API exists, people can start writing their own extensions that can possibly be downloaded and used by others too.

Basically we often try to compare it with systems like Google Desktop Search: Google provides the basic infrastructure for desktop indexing and also provides some indexers, like for Office and PDF files. However, they also set up an API that allows people to add functionality.

1.3 Lessons learned

The lessons we learned from the development of the first framework can be summarized in 4 main points:

- The first version of AMG internally used the Ariadne application profile of LOM. The use of AMG within environments like the submission workflow of DSpace, showed the limitations of this approach: we need full LOM support instead of limiting ourselves to a certain application profile of LOM. We tackled this issue by using the LOM Java API from now on, which is a Java implementation for dealing with LOM metadata [4].
- The difference between using AMG in standalone versus web service mode was not clear. Therefore, in the new version of AMG we wanted to make a clear distinction between the several modules:
    - The actual "core" functionality, which deals with determining values for the metadata fields.
    - Wrappers around the core:
        - Standalone classes for using AMG in standalone mode.
        - Web service server-side skeletons and client-side stubs that wrap the core metadata generating functionality into web service calls. This module serializes and deserializes SOAP data to platform-specific data objects, like Java objects.
    - Clients that want to use the framework will then be able to use the standalone or web service wrapper.
- For the first version of AMG we started with some Java classes that were then turned into a web service, by using the code generating functionality of the particular web service platform we used (Axis 1.2) [5-7]. This made our code non-interoperable with clients that use other web service platforms. This is unacceptable for us because interoperability is important, at two levels:

  - We want to provide the option to create an AMG implementation in the platform of your choice. For now we have been using Java implementations, but we also foresee an AMG installation that for example uses .NET for handling MS Office files [8].
  - It should be possible to write the client side code in the platform of your choice.
- The terminology we used seemed somewhat confusing. From now on we will consistently use "automatic metadata generation" for the work we are doing.

1.4 New version of the framework

To accommodate for the lessons we learned from the first design (see 1.3), we started our re-design by first thinking about AMG at a more abstract level, specifying what operations should be offered by an AMG installation, i.e. a software system that offers some form of automatic metadata generation. This is captured in what we call the "Simple AMG Interface", SAmgI. This abstract specification is then syntactically defined by binding it to WSDL. This newly outlined API finally results in web services and standalone bindings for different platforms.

1.4.1 The Simple AMG Interface: an abstract specification for AMG services

The Simple AMG Interface includes four groups of operations. For sake of completeness, we should tell that it actually includes a fifth group that deals with the authentication and session management. However, because these operations are fully based on the same operations in the Simple Query Interface [9], we don't mention them here.

((1)) Because different applications need metadata in different formats (e.g. LOM, DC-XML, DC-RDF, MPEG-7, …) we want to let clients specify what kind of metadata they want to retrieve. Therefore we provide operations for defining the metadata language/format/schema/standard to use.

((2)) A second group defines operations related to the conflict handling method to use. In [1], we explain "conflict handling" in the context of automatic metadata generation. The idea is that parts of the global AMG framework each generate values for some of the metadata fields. Because those parts can generate different values for the same metadata fields, conflicts can occur in the generated values. This is why we created the idea of conflict handling:

a) First of all, we need to keep track of information that can be used to solve conflicts. This is what we call the "merging information". In the concrete implementation that we developed so far, we use confidence values for this purpose. Confidence values represent how sure the generator of the metadata value was, and as such represent the measure of confidence or certainty for the specific value for the metadata field. Of course, even when we have the merging information, there are still several possible options. For multi-valued metadata fields one could for example take the option to just choose all values, no matter how good their merging information is. Another option would be to just take 1 of the best ones. A third option would be to take all the best ones. This idea of looking at the merging information and deciding what to do with it is captured in the notion of "conflict handling".

b) Because in some cases the type of conflict handling depends on the type of merging information, we decided to glue them together into what we call the "conflict handling strategy/method", which is just a combination of type of merging information plus type of conflict handling. We for example have developed a ConfidenceValues-TakeHighestAll strategy.
In the SAmgI specification, a conflict handling strategy is identified by a case-insensitive String. We provide an operation that allows retrieving a list of all possible conflict handling strategies that are supported by the service endpoint.

((3)) A third group of operations deals with the MetadatasourceIds. As explained shortly in [1], a MetadatasourceId identifies the learning object, or the context that a learning object resides in. Based on the MetadatasourceId, the AMG framework has enough information to do its metadata generation job. An example is the OCWMetadatasourceId that identifies the "OpenCourseWare" context of an OCW document. Concretely, this identifier is just the URL location of the OCW document, as from that URL we can fully identify the OCW document.
We include the option to retrieve all supported MetadatasourceIds. This is important because not all SAmgI implementations will support all MetadatasourceIds.

((4)) The last group contains the operations that are related to the actual metadata generation. It will allow a client to feed his MetadatasourceIds and ask the SAmgI installation to generate metadata for it.

In Table 1 we summarize all operations that are part of the SAmgI specification at this moment. The number between double brackets in the left column corresponds to the numbers in the text above.

Authentication and Session Management	
createSession	Creates a session
createAnonymousSession	Creates an anonymous session (without requiring an account at the system where the session will be created)
destroySession	Destroys a session

Simple AMG Interface		
((1))	setMetadataFormat	Sets the metadata format that will be used for the generated metadata
	getMetadataFormat	Gets the current metadata format
	getSupportedMetadataFormats	Retrieves all supported metadata formats
((2))	setConflictHandlingMethod	Sets the method that is used to solve conflicts in the generated metadata
	getConflictHandlingMethod	Gets the current conflict handling method
	getSupportedConflictHandlingMethods	Retrieves all supported conflict handling methods

((3))	getSupportedMetadatasourceIds	Retrieves a list with the names of the supported MetadatasourceIds.
	getSupportedMetadatasourceIdsSchema	The same as getSupportedMetadatasourceIds, but now returning the XML schema describing the supported MetadatasourceIds.
((4))	getMetadata	Generates and returns metadata for a given learning object
	getMetadataWithMergingInformation	Generates and returns metadata together with merging information for a given learning object (together this makes up what we call AmgMetadata)
	convertMetadata	Converts an AmgMetadata instance to another metadata format

Table 1: overview of all SAmgI operations

A complete description of the specification, including more information about each of the operations can be found in our specification document [10]. We have created a first version of the specification in collaboration with other people, and are using the result in some first implementations. We are however continuously gathering feedback on the specification, and the readers are encouraged to contact the authors to contribute or comment on it.

1.4.2 SAmgI syntactically specified by XML schemas and WSDL

To allow interoperability of different SAmgI installations, it is important that the data formats of interchanged messages are specified in an interoperable way. We did this by specifying XML schemas for the data formats. One of the most important schemas is the one we defined for the exchange of generated metadata. For this exchanged metadata we could not just rely on the existing XML schemas of for example LOM because:

- in the design of SAmgI we do not want to restrict ourselves to one particular metadata format.
- besides from the metadata itself, we also need extra information about the generated values. More specifically the previous ideas of merging information and conflict handling should be captured in the schema in some way.

Below we will outline some ideas that drove the creation of this data format for the exchanged metadata, and an example of how it looks like.

During the development of the schema for the exchanged metadata, we had several design goals in mind:

- it should allow having a complete history of the metadata generation process. For each metadata field, it should be possible to see all values that were generated throughout the process, and which values were replaced by which other values. For example, if one metadata generator first determined the author to be personX, and another one replaces that by personY, it should be visible in the developed format.

- we did not want to commit ourselves to a particular metadata standard. Therefore the developed schema cannot make a reference to a particular metadata format. However bear in mind the difference between the specification itself, which does not commit to a particular metadata standard versus a particular SAmgI installation, which will choose a metadata format for internal use. In our case, our particular implementation will use LOM as the (internally) used metadata format.
  This is much like in SQI [9], where the specification does not impose a query language or results format. However, for a client to be able to call the SQI target, he must know the query and results format.

Code sample 1 shows an example of what the exchanged metadata looks like. We can distinguish the following important parts:

((1)) metadataString: this is just a string serialization of the metadata in the used metadata standard. In the implementation that we made, this will be a string representing the LOM XML metadata.

((2)) mi: this represents the merging information for the metadata instance, containing miItem child elements. Each of them corresponds to a metadata field, like LOM.General.Language and it contains:

a) the fieldname
b) the history of chosen values for the field (fieldMi), containing:
   i) the currently valid group of values for the metadata field (fieldValuesGroup, containing fieldValuesGroupElements)
   ii) the previously valid merging information (previousMI)
   As you can see, we use recursion for this. Each fieldMI contains the currently valid group of elements and the previous fieldMI. This way it should be easy to retrieve previous values for a metadata field.
c) the values that were never chosen, because they were never better than existing ones (unchosenValuesGroup)

```
      <?xml version='1.0' encoding='UTF-8'?>
      <amgMetadataElement ...>
         <metadataString>
((1))       &lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;lom&gt;
      &lt;general&gt; &lt;title&gt; &lt;string language="en"&gt;Example
      Title&lt;/string&gt; &lt;/title&gt; &lt;/general&gt; ...
         </metadataString>

         <mi>
           <miItem>
a)           <fieldName>LOM.General.Title</fieldName>
b)           <fieldMi>
               <fieldValuesGroup>
```

((2))

c)

```
               <fieldValuesGroupElement>
                  <fieldValue>
                     &lt;string language="en"&gt;Example
Title&lt;/string&gt;
                  </fieldValue>
                  <miValue>0.8</miValue>
                  <valueGenerator>WordGenerator</valueGenerator>
               </fieldValuesGroupElement>
            </fieldValuesGroup>
            <previousMI>
               <fieldValuesGroup>
                  <fieldValuesGroupElement>
                     <fieldValue>
                       &lt;string language="en"&gt;Example old
title&lt;/string&gt;
                     </fieldValue>
                     <miValue>0</miValue>
               <valueGenerator>OfficeGenerator</valueGenerator>
                  </fieldValuesGroupElement>
               </fieldValuesGroup>
            </previousMI>
         </fieldMi>
         <unchosenValuesGroup/>
         </miItem>
     </mi>
</amgMetadataElement>
```

Code sample 1: XML instance representing the exchanged metadata format

The developed XML schemas for the exchanged data formats should be used by all SAmgI installations, both standalone implementations and web service implementations. To allow interoperability between web service implementations (both the services and the clients), we also developed a WSDL schema that doesn't include any web service platform specific data types.

1.4.3 SAmgI-WSDL bound to concrete implementations

The abstract specification, the developed XML schemas and the WSDL are finally bound to concrete implementations. At the moment we are developing Java implementations, both standalone ones and web services ones. For the web service one, we plan to do a version for Axis1 [5], Axis2 [6] and the Microsoft Web services [7].

2 How to do automatic metadata generation for your own collection of documents

In order to implement SAmgI for a system or collection of documents, there are basically three options, depending on the requirements and the characteristics of the system, like the access to it and the platform is it written for.

Option 1 is to just act as a client of some existing SAmgI implementation. This would come down to using it more or less like a black box, asking the system for metadata for a certain object. Because in this case the metadata generation is not at all fine-tuned for the particular case, the result would be a rather small set of metadata.

Option 2 is to create from scratch a complete SAmgI installation, potentially reusing existing components. In this case an implementation is created that conforms to the abstract SAmgI specification, the XML schemas for the data types, and the WSDL (in case it is a web service implementation).

Option 3 allows making an easier and faster implementation of SAmgI for your system. It comes down to writing a small layer on top of the existing system or collection of documents, that can be called by existing SAmgI implementations like the Java version we are developing. This layer should allow retrieving the properties of the learning objects, like the author, the title, or anything else that is available within the system. In a next step a ContextBasedGenerator will be created, within an existing SAmgI implementation that will use the offered layer to build the metadata.

3 Case study: indexing ProLearn material

3.1 AMG for the ProLearn deliverables

As the first big case study of the new AMG framework, we have been developing a system that indexes all material that is produced in the context of ProLearn Network of Excellence on Professional Learning [11]. To manage all ProLearn documents a shared workspace system is used, called the Agora Groupware Web Server [12]. In a first step we focus on the deliverables that are produced within ProLearn. In a next step, also the other material like the papers is processed.

To generate the metadata we chose Option 3 of section 3, which came down to writing a web services layer on top of the AGWS system. Our particular SAmgI implementation was then extended with a module that uses those web services for generating the metadata for the ProLearn deliverables. This is represented by the upper rectangle in Figure 1.

3.2 What to do next with the generated metadata: allowing search and retrieval

Although the generation of metadata for a learning object is the key focus of our work so far, we also want to make use of that metadata afterwards, for example for searching. To create this search functionality we wrote an extension of AMG using the combination of a Lucene index [13], and the Simple Query Interface (SQI) [14].

Lucene is developed by the Apache group, and provides a high-performance, full-featured text search engine library, that we use for storing the generated metadata.

SQI is a definition of web services that enable querying Learning Object Repositories in a standardized way [9]. In our case, it defines the query interface that we will use for searching the generated metadata.

This is all represented by the bottom rectangle in Figure 1.

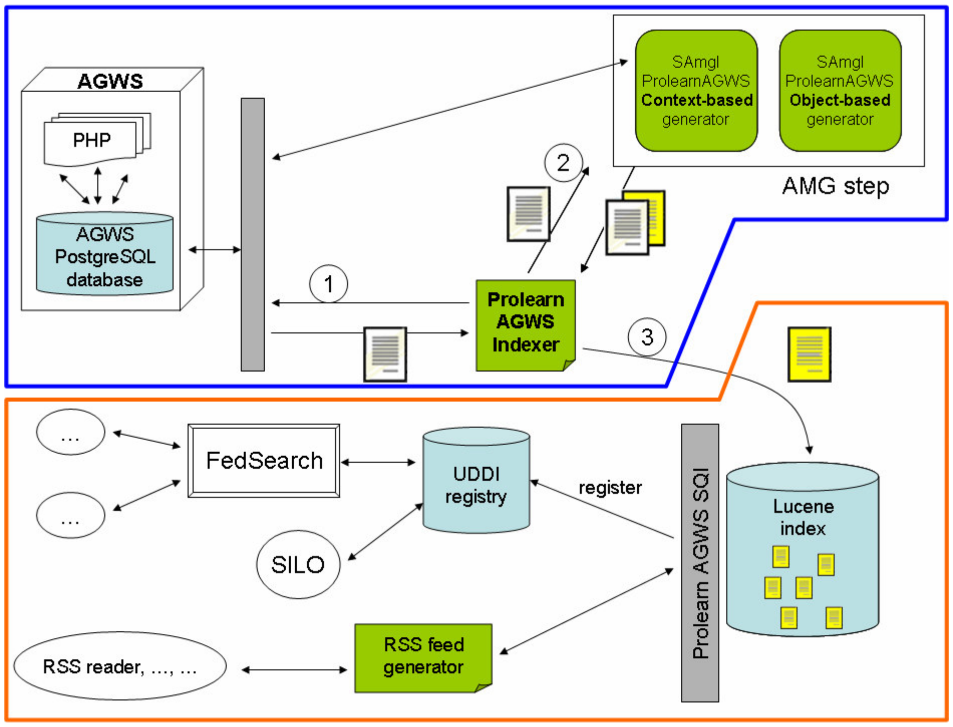

Figure 1: overview of how we index the material of the ProLearn Network of Excellence. The same figure can be used for material of other systems that needs to be indexed.

4 Evaluation of the automatically generated metadata for ProLearn deliverables

4.1 Experimental setup

In order to evaluate the relative quality of the metadata records generated by AMG for the ProLearn deliverables, we set up an experiment to compare them with existing human generated records present in the ARIADNE repository [15]. During the experiment several reviewers had to grade the quality of a set of records sampled from both sources. As the universe of manual records, we selected metadata records about Information Technologies objects that were available in English inside the ARIADNE repository. From this universe (425 records) we randomly selected 10. The universe of automatic records was the 114 records generated with the SAmgI framework for the ProLearn deliverable documents. From this universe we also randomly sampled 10 records.

Following a common practice to reduce the subjectivity in the evaluation of the quality of metadata, we used an evaluation framework. The selected framework was the one proposed by Bruce and Hillman [16]. It was selected because it summarizes the quality of the metadata record in 7 easy to understand and measurable parameters: completeness, accuracy, provenance, conformance to expectation, logical consistency and coherence, timeliness and accessibility. All participants in the experiment were requested to read the definition of each parameter before grading the records. The definitions were also available during the evaluation process.

The experiment was carried out online using a web application [17]. The user logs in into the system with his name. The system presents him or her with the instructions containing the explanation of the evaluation framework. After reading the instructions, the user is presented with a list of the 20 selected objects in no specific order. When the user selects an object, a representation of its LOM record is displayed. The user can then download the referred object for inspection. Once the user has reviewed the record and the objects, he is asked to give grades in a 7-point scale (From "Extremely low quality" to "Extremely high quality") for each one of the 7 parameters. Only participants that grade all the objects were considered in the experiment.

The experiment was available for 2 weeks. During that time 33 different participants entered the system, but only 22 of them completed successfully the review of all the 20 objects. From those 22, 17 (77%) work with metadata as part of their study/research activities; 11 (50%) were undergraduate students in their last years, 9 (41%) were postgraduate students and 2 (9%) had a Ph.D. degree. All of them were in full capacity to understand the nature and meaning of the examined objects and their metadata records. The reviews given by those 22 participants were the ones considered in this study.

4.2 Data analysis

Because of the inherent subjectivity in measuring quality, the first step in the analysis of the data was to estimate the reliability of the evaluation. In this kind of experiment, the evaluation could be considered reliable if the variability between the grades given by different reviewers to a record is significantly smaller than the variability between the average grades given to different objects. To estimate this difference we use the Intra-Class Correlation (ICC) coefficient [18] which is commonly used to measure the inter-rater reliability. We calculate the average measure of ICC using the two-way mixed model, given that all the reviewers grade the same sample of objects. In this configuration, the ICC is equivalent to another widely used reliability measure, the Cronbach's alpha. The results for each quality parameter are reported in the Table 2.

Parameter	ICC (average, two-way mixed) ≡ α
Completeness	0,881
Accuracy	0,847
Provenance	0,701
Conformance	0,912
Consistency & Coherence	0,794
Timeliness	0,670
Accessibility	0,819

Table 2: Intra-Class Correlation (ICC) coefficient for measuring the reliability of the evaluation

The only value that falls below the 0.7 cut-off value to be considered acceptable is the Timeliness parameter. In other words, the reviewers did not "agree" in the measurement of the timeliness. For the other parameters, the ICC suggest that the reviewers provided similar values and further statistical analysis could be performed.

The second step is to asses if there is a difference between the average grade given to automatically generated records and the average grade given to manual generated records. These averages values are presented in Figure 2. To statistically establish whether the difference between average values is real or a by-product of the natural variance, we proceed to apply a one-way ANOVA test. Our null hypothesis is that there is no difference between the grades given to automated and manual records. Our alternative hypothesis is that there is indeed a difference. The results are presented in Table 3. All results where obtained with an F(1,18) distribution.

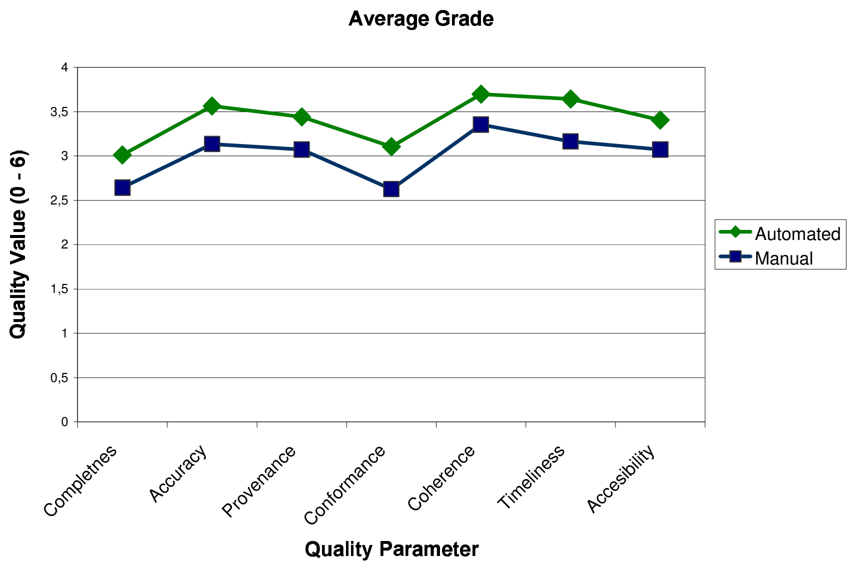

Figure 2: Average quality grade for the different parameters

Parameter	F-value	Significance (two-tailed)
Completeness	2,286	0,148
Accuracy	3,640	0,073
Provenance	5,060	0,037
Conformance	2,420	0,137
Consistency & Coherence	4,345	0,052
Timeliness	16,811	0,001
Accessibility	2,727	0,112

Table 3: significance of the difference between the given grades

While the automatically generated records seem to be, in average, graded 0.4 points higher than the manual generated records, in most of the parameters (completeness, accuracy, conformance to expectations, consistency & coherence and accessibility) we cannot reject the null hypothesis: the difference found is just consequence of the random variability. The significant difference found in provenance value could be explained as that all the automated records had the same origin, ProLearn deliverables, but cannot be generalized to other sources. While the timeliness parameter also shows a significant difference, the low value of reliability of this measure prevents us to draw conclusions from it.

4.3 Evaluation conclusions

There is no statistical difference between the quality grades given to a random sample of ARIADNE records and the ones produced automatically by SAmgI. That means that for the reviewers their quality is equivalent. We can introduce the automatically generated records into ARIADNE without degrading the quality of the repository. These results could not be generalized to any kind of human generated metadata or any kind of automatically generated metadata. This evaluation only holds between ARIADNE metadata records and the ProLearn deliverables records.

In the future we plan to analyze the quality evaluation in more detail, studying the interrelations between different quality parameters. Also, we will try to create an automated quality evaluator to avoid the need of evaluations for each new automatic metadata generator.

5 Future work

For the core metadata generation functionality, we incorporate existing implementations of relevant information retrieval. Examples of things we have integrated so far are the ngram-based language determination [19] and the extraction of MS Office properties, using Jakarta POI [20] and PDF properties, using PDFBox [21]. A next step will be to incorporate techniques for keyword extraction, like KEA [22] and GATE [23].

Projects related to AMG are AMeGA [24], which does more conceptual work on what the role of automatic metadata generation can be throughout the complete metadata generation or submission process. In the future we will keep on looking for related work, like the Automatic Metadata Extractor [25] and the Automatic RDF Metadata Generator [26].

Furthermore we will are working towards a version 1.0 of the SAmgI specification and its implementations.

On a longer term, we plan to also look at newer extensions to AMG, e.g. taking into account the complete lifecycle of the learning object as a possible source of metadata.

6 References

1. Cardinaels, K., M. Meire, and E. Duval. *Automating Metadata Generation: the Simple Indexing Interface*. In *International World Wide Web Conference (WWW)*, 2005, Chiba, Japan: International World Wide Web Conference Committee (IW3C2). http://ariadne.cs.kuleuven.ac.be/amg/publications.php.
2. Ochoa, X., et al. *Frameworks for the Automatic Indexation of Learning Management Systems Content into Learning Object Repositories*. In *ED-MEDIA World Conference on Educational Multimedia, Hypermedia & Telecommunications*, 2005, Montreal, Canada, Education & Information Technology Library.

http://ariadne.cs.kuleuven.ac.be/amg/publications.php and http://www.editlib.org/index.cfm?fuseaction=Reader.ViewAbstract&paper_id=20276.

3. Meire, M., *Homepage of AMG (Automatic Metadata Generation)*, http://ariadne.cs.kuleuven.ac.be/amg.
4. Hubick, C., *Learning Object Metadata (LOM) Java API*, http://sourceforge.net/projects/lom-j/, http://sourceforge.net/projects/mime-dir-j/ and http://sourceforge.net/projects/vdex-j/.
5. Apache Software Foundation (ASF), *Axis 1*, http://ws.apache.org/axis.
6. Apache Software Foundation (ASF), *Axis 2*, http://ws.apache.org/axis2.
7. Microsoft, *Web Services Enhancements*, http://msdn.microsoft.com/webservices/webservices/building/wse.
8. Verbert, K., *Homepage of ALOCoM*, http://ariadne.cs.kuleuven.ac.be/alocom.
9. Simon, B., et al. *A Simple Query Interface for Interoperable Learning Repositories*. In *Workshop on Interoperability of Web-Based Educational Systems in conjunction with 14th International World Wide Web Conference (WWW)*, 2005, Chiba, Japan: International World Wide Web Conference Committee (IW3C2)
10. Meire, M., K. Cardinaels, and E. Duval, *Simple AMG Interface (draft specification)*. 2006, http://ariadne.cs.kuleuven.ac.be/wordpress/amg/index.php/2005/09/21/abstract-specification-for-services-that-offer-automatic-metadata-generation/.
11. Prolearn, *ProLearn Network of Excellence on professional learning*, http://www.prolearn-eu.org/.
12. Agora Systems S.A., *Agora Groupware Web Server (AGWS)*, https://agws.dit.upm.es/enter.php.
13. Apache Software Foundation (ASF), *Lucene*, http://lucene.apache.org/java/docs/index.html.
14. *Simple Query Interface (SQI)*, http://prolearn-project.org/lori.
15. Ariadne foundation for the European Knowledge Pool, *Ariadne repository*, http://www.ariadne-eu.org/.
16. Bruce, T.R. and D.I. Hillmann. *The Continuum of Metadata Quality: Defining, Expressing, Exploiting*, 2004. In D. Hillmann and L. Westbrooks, Metadata in Practice (Chicago: American Library Association)
17. Ochoa, X., *Evaluation experiment for comparing the quality of automatically versus humanly generated metadata.* 2006, http://ariadne.cti.espol.edu.ec/Metrics/instructions.jsp.
18. Shrout, P.E. and J.L. Fleiss, *Intraclass Correlations: Uses in Assessing Rater Reliability.* Psychological Bulletin, 1979(2): p. 420-428
19. Canvar, W.B. and J.M. Trenkle, *N-GRAM-Based Text Cathegorization*, http://www.nonlineardynamics.com/trenkle/papers/sdair-94-bc.ps.gz.
20. Apache Software Foundation (ASF), *Jakarta POI 2005: Java API To Access Microsoft Format Files: http://jakarta.apache.org/poi/*,
21. *PDFBox*, http://www.pdfbox.org/.
22. *KEA automatic keyphrase extraction*, http://www.nzdl.org/Kea.
23. *GATE General Architecture for Text Engineering*, http://gate.ac.uk/.
24. Greenberg, J., *AMeGA (Automatic Metadata Generation Applications) project*, http://ils.unc.edu/mrc/amega.html.

25. *Automatic Metadata Extractor*, http://epsilon.uwaterloo.ca/TextMiner/MetadataExtractor.aspx.
26. *Automatic RDF Metadata Generator*, http://www.scit.wlv.ac.uk/~ex1253/metadata.html.