Linked Open Webble: Connecting Webbles to the World Wide Web

Nicolas Spyratos and Tsuyoshi Sugibuchi

Laboratoire de Recherche en Informatique, Université Paris-Sud 11, France
{Nicolas.Spyratos,Tsuyoshi.Sugibuchi}@lri.fr

Abstract. Webble technology is a media technology allowing media generation by direct manipulation. Webble users can build systems and interfaces by interactively combining visual components, called webbles, and dynamically connecting functions of components to create new, composite components. A casual user can simply place one webble on another and plug it in, and the combination will be able to run immediately *assuming* the connected functions are compatible. However, the current webble technology provides virtually no means for (a) searching a potentially huge pool of webbles to find those needed for a specific application and (b) knowing whether the functions of two webbles are compatible (and therefore whether the webbles can be connected). In this paper, we propose an approach to a solution for this problem.

1 Introduction

Webble technology is a media technology allowing media generation by direct manipulation [1]. Webble users can build systems and interfaces by interactively combining visual components, called *webbles*, and dynamically connecting functions of components to create new, composite components. Each function of a webble is physically represented as a webble slot. Casual users can simply place one webble over another (by drag-and-drop), plug it in, and the combination will be able to run immediately, *assuming* that the connected functions are compatible.

Therefore, when a webble is dropped on another, one of two things can happen: if the relevant slot types of the two webbles are compatible then the webbles are effectively connected and the new, composite webble is available for further use; otherwise there is no connection established and therefore no creation of a new, composite webble. In other words, placing one webble on another and plugging it in is *not* necessarily creating a connection, unless there is slot compatibility.

The long term vision of webble technology is to start with an open pool of atomic webbles created by an initial group of technology oriented users, and grow it to a world wide webble pool that can be accessed freely by casual or professional users. The webble pool users will either use existing components for their applications, or create new components from old and contribute to growing the pool by making their webbles available to the webble community.

O. Arnold et al. (Eds.): WWS 2013, CCIS 372, pp. 66–78, 2013.

However, in the current version of webble technology, there is little or no help provided to the casual user in order to (a) search a potentially huge pool of webbles to find those needed for a specific application and (b) know whether the functions of two webbles are compatible (and therefore whether the webbles can be connected). As a consequence the user must try to connect webbles by trial and error - a formidable task, as the number of slot combinations is huge even in a pool of a few tens of webbles.

In this paper, we propose to solve this problem by introducing annotations at two levels: at the webble level and at the slot level. Our claim is that if these annotations are based on standard, controlled vocabularies then one can define search mechanisms (based on those vocabularies) allowing to access webbles of interest, and to have information about compatibility of their slots. Additionally, the world of webbles can then communicate with other worlds using the same vocabularies, and in particular it can communicate with the world wide web.

Today, there are several standard vocabularies, such as Dublin Core[2], IEEE's LOM [15], CIDOC-CRM[3], ACM's CCS [16] and others. These vocabularies are rather widely used today in several areas, including digital libraries, archives [4], or distant learning systems, to mention just a few areas. Annotation framework using such standard vocabularies or more sophisticated ontology representing domain knowledge [5][6][7] is continuously very hot research topic and already in use in various application domains. One of most successful examples is semantic annotation of biomedical data. Nowadays various biomedical data are annotated by using several standard vocabularies [8][9] and those annotations are widely used for biomedical data analysis [10] and information retrieval[11].

Our proposal is also motivated by another simple observation regarding webble technology. To begin with, from a programming point of view, webbles can be regarded as a visual, dynamically typed programming language. Compared to static typing, dynamic typing has the advantage of enabling the connection of arbitrary pairs of slots during runtime. However, such over-flexibility entails the risk of misconnections, as well as difficulties in choosing an appropriate combination of slots from a huge number of possible combinations.

One possible solution is to enhance dynamic programming with *optional type annotation*[12][13][14]. It is the kind of annotation that programmers optionally embed into source code for indicating types of variables or function signatures. Optional type annotation has no effect on the run-time semantics of the programming language. However, it can be used for improving productivity and runtime performance of dynamic programming languages by static type checking before execution, JIT (just-in-time) compiler optimization and content suggestion in IDEs (integrated development environments). Today, optional typing is a very popular approach for introducing more stability and strictness to dynamic programming languages while preserving their flexibility.

The idea of optional type annotation is not limited to the programming world. In the world wide web, we also find various *microformats* for annotating pieces of HTML documents by using predefined, controlled vocabularies. For instance, the *hCard* standard defines a set of class names for indicating contact

information. By using hCard, we can explicitly indicate telephone numbers like `<span class="tel">0678901234</span>`. Microformats also enable us to preserve expressiveness of HTML itself and to *optionally* introduce semantics of data in HTML documents.

From these observations, we believe that there can be various useful applications of optional type annotation for webbles as well. However, we have to keep in mind that webbles is *not* a RAD (Rapid Application Development) tool. It is a media architecture. Therefore, we should not limit possible annotation types for webbles only to those used for data types in programming.

For instance, if you make a dialog webble for entering book reference data, it is a good idea to annotate both the webble and its slots, using the Dublin Core vocabulary. In this way, if later on you develop a new application, say for EPUB books, you can easily find that dialog from a webble repository and almost automatically connect it to your new application. This is possible because Dublin Core is not only a programming technology but also a common vocabulary used for annotating files of various media formats, including EPUB. We can also imagine similar examples not only for Dublin Core, but also DBpedia, YAGO, AAT, MeSH and any other controlled vocabularies used in the Web. By annotating webbles with such widely used vocabularies, webbles can gradually become citizens of the Linked Open Data (LOD) world.

In this paper, we introduce a generic and formal framework for annotating webbles. The kind of vocabulary that we envisage can be a pre-defined vocabulary specific for the webble framework, or the LOD world. The framework that we propose here works with any kind of controlled vocabulary. Moreover, our framework doesn't touch the webble architecture itself - it just overlays an additional, semantic layer on top of the naked webble architecture.

2 Webble Annotations

The current webble technology offers some means to attach annotations both at webble level and at slot level. At webble level we find several kinds of annotations in the current webble implementation: name, class name, developper, group identifiers, description keywords and "metadata". However, the terms in these annotations do not come from any standard vocabulary and the mechanism for creating metadata is rarely used (i.e. most webbles have empty metadata).

At slot level, we find two kinds of annotation: (a) a slot name and (b) a slot type. However, none of these information items is exploited from the user's point of view. Moreover, there is no mechanism available to express attributes (or properties) of a slot other than its name and type.

Another problem is that the names in these annotations do not come from any standard vocabulary either. For instance, if someone wants to search webbles by name, he has to read a document about webble file format to identify the name of the annotation which represents webble names. This lack of relationship with widely-used controlled vocabularies prevents visibility, searchability, and reusability of webbles in the context of the world-wide-web.

To solve this problem, we propose to enrich the existing annotations of a webble at both levels (i.e. at webble level and at slot level), and moreover use annotations for designing search mechanisms and for reasoning about webbles. The annotation framework that we propose in this paper aims at the following goals:

Easy to define. To encourage webble developers and users to start annotating webbles, we should keep the annotation model as simple as possible. From user's point of view, a webble annotation in our framework is just a set of attribute-value pairs. Our framework also provides a controlled vocabulary for webble annotation: it's a small set of terms borrowed from RDF together with a few webble-specific terms. Reasoning about annotations is based on a set of entailment rules defined by the framework. However, as far as users are concerned, reasoning is transparent: it is basically a kind of detail users do not need to care about.

Annotation of both media containers and media contents. Webble is not only a GUI toolkit, but also a media architecture. A webble can be a container of media contents. When we annotate webbles, we need to distinguish between media containers and media contents. For example, suppose an image webble has an annotation "`dc:creator` is Hokkaido university". In this case, Hokkaido university is a creator of what? The webble itself, or the image displayed by the webble? Actually, we need to handle both cases. Out framework allows users to annotate both webbles and contents represented by webbles.

Cooperation with webble architecture. Our framework can cooperate with the basic mechanisms of webbles, in particular, slot connection and webble composition. It can map slot values to annotations, and it can reify annotations as webbles. These features allow us to interactively annotate webbles and media contents by using slot connections and webble compositions.

Easy to map to RDF. After annotating webbles, the next step is to make webbles visible in the LOD world. A straightforward way to do this is to represent webble annotations by using the standard metadata format in LOD world, namely RDF. Our framework provides a simple mechanism to map webble annotations to RDF graphs. From webble annotations, our framework produces RDF graphs representing annotations of webbles, annotations of contents, and the relationship between a webble and its contents.

Enhance the behavior of webbles. If the webbles are annotated, then the webble runtime environment can use annotations for various use. In particular, RDF mapping of webble annotations enables us to use the powerful reasoning system of RDF in order to control the behavior of webbles. In this paper, we demonstrate rule-based restrictions of webble composition and slot connection by using webble annotations.

2.1 Annotating Webbles and Their Slots

Figure 1 shows the work flow in an example that we shall use as a running example throughout this paper. In this example, a user wants to create an image

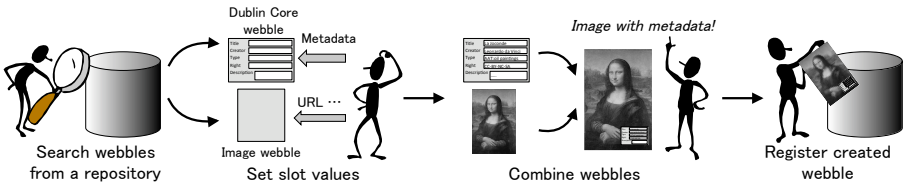

Fig. 1. A example work flow

webble whose content (i.e. the image contained in the webble) is annotated by the content of a metadata webble. To achieve this goal, the user has to do the following: (1) search for webbles in repositories for creating the composite webble satisfying his goal; (2) set slot values, define webble compositions and slot connections to create the intended webble; and (3) possibly store the created webble into a repository for future reuse. Figure 2 shows the structure of the composite webble.

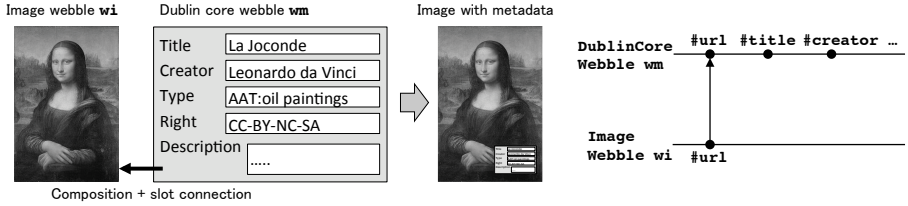

Fig. 2. Structure of the example webbles

In the composite webble, we can see an image webble `wi` which can contain and display an image, and a metadata webble `wm` which can contain metadata defined in Dublin Core (hence its name "DublinCoreWebble"). The user wants to compose these two webbles so that the metadata contained in `wm` is added to the image displayed by `wi`.

This work flow can be done perfectly without any annotations at all. However, from the practical point of view there are several problems.

1. How can the user efficiently search for webbles in a webble repository?
2. How can the user make this composition without tedious try-end-error repetition?
3. How can the user expose results of the webble composition in a searchable form for future reuse?

To solve these problems, we need (1) well-organized webble annotations on which to base search mechanisms (2) a query language for searching webble repositiories (3) a mechanism preventing users from performing forbidden operations, and (4) a mapping rule from annotations of composite webbles to a searchable form.

In this paper, we first discuss webble annotations (point (1) above); then a mapping rule from webble annotations to RDF (point (4) above); and finally we demonstrate a plug-in type system to restrict user's operations in order to avoid errors (point (3) above). We do not address the issue of defining a query language (point (2) above). To begin with, let's define formally what an annotation is.

Definition 1 (Webble annotation and slot annotation). *Let i be the identifier of a webble or of a slot. The annotation of i, denoted An(i), is defined to be a set of attribute-value pairs, namely:* $An(i) = \{(A_1, v_1), ..., (A_n, v_n)\}$.

There are several important points to mention regarding the definition of a webble annotation. Firstly, we assume that both webbles and slots are associated with identifiers, and moreover that we can tell whether an identifier is a webble identifier or a slot identifier (e.g. from its syntax). In the rest of this paper, we use the following conventions: (1) if w denotes a webble then `w` denotes the identifier of w, and (2) if s is a slot of webble w then `w:#s` denotes the identifier of slot s of w.

Although we do not discuss how webble annotations are defined, we assume two types of annotations: *system annotations*, defined by webble developers and *user annotations* defined by users. Attributes and values of system annotations are pre-defined by developers or dynamically assigned by webble runtime environments, while user annotations are defined by users who wish to append application-specific inforation to webbles.

Finally, we assume that all attribute names $A_1 \ldots A_n$ come from some controlled vocabulary. In particular, in order to be able to map webble annotations to RDF, we will introduce a restriction such that every attribute name must be the URL of an RDF property. Therefore in the rest of the paper we use only URLs of RDF properties as attribute names.

Figure 3 shows an example annotation of an image webble *wi*. Basically all annotations in this example are system annotations.

```
     An(wi) = {(dc:title, "Image Webble"), (rdf:type, ImageWebble),
               (dc:creator: "Hokkaido Univ."), (wb:hasSlot, wi:#url)}
An(wi:#url) = {(dc:title, "URL"), (rdf:type, TextSlot),
               (wb:hasValue, http://a/b/c.jpeg)}
```

Fig. 3. Webble annotation example

In this example, some basic information (name, title, ...) are described by using terms from Dublin Core, whose name space is denoted as `dc`. The example annotation also describes ownership and value of `url` slot by using `wb:hasSlot` and `wb:hasValue` that come from a controlled vocabulary that we have specifically designed for webble annotations. We call this vocabulary *webble vocabulary* and in the rest of the paper its name space is denoted as `wb`. Note that `wb:hasValue` is a typical dynamic annotation. Indeed, the value of this annotation is dynamically assigned by a webble runtime environment to indicate the slot value of an annotation.

An important point to notice here is that webble annotations describe only webbles and their slots. Indeed, the example annotation of Figure 3 does not say anything about the image `http://a/b/c.jpeg` displayed by the image webble `wi`. In other words, webble annotations in this figure annotate only media *containers* and do not annotate media *contents*.

2.2 Annotating Slot Values

To annotate media contents, our approach uses an entailment rule to *infer* annotations of media contents from annotations of media containers and their state. In other words, our framework assumes that metadata of a media content is represented as state of a media container. Regarding the webble architecture, the state of a webble is externalized as a set of slot values.

```
        An(wm) = {(dc:title, "Metadata"), (rdf:type, DublinCoreWebble),
                  (wb:hasSlot, wm:#url), (wb:hasSlot, wm:#title), ...}
   An(wm:#url) = {(dc:title, "URL"), (rdf:type, TextSlot),
                  (wb:hasValue, "") }
 An(wm:#title) = {(dc:title, "Title"), (rdf:type, TextSlot)}
                  (wb:represents, dc:title), (wb:annotatesValueOf, wm:#url),
                  (wb:hasValue, "La Joconde") }
            ... ... ...
```

Fig. 4. Slot value annotation example

Figure 4 shows an example annotation of DublinCoreWebble `wm` and its slots (only annotations of `url` slot and `title` slot are shown).

The properties `wb:represents` and `wb:annotatesValueOf` in the annotation of `title` slot are terms from the webble vocabulary used for annotating *values* of slots. Intuitively, the annotation of `title` slot says that "the value of `title` slot represents `dc:title` of something specified by `url` slot".

This statement is represented by three terms of webble vocabulary and one entailment rule. Figure 5 illustrates the mechanism of slot value annotation. In this figure, `wb:represents` associates slot t with an attribute name p.

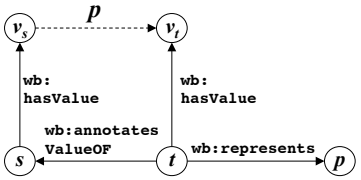

Fig. 5. `wb:represents` and `wb:annotatesValueOf`

`wb:annotatesValueOf` associates slot t with slot s. The slots s and t have values v_s and v_t. For these annotations, webble vocabulary defines the following entailment rule R_1.

$$\frac{(\texttt{wb:hasValue}, v_t) \in \texttt{An}(t) \wedge (\texttt{wb:represents}, p) \in \texttt{An}(t) \wedge (\texttt{wb:annotatesValueOf}, s) \in \texttt{An}(t) \wedge (\texttt{wb:hasValue}, v_s) \in \texttt{An}(s)}{(p, v_t) \in \texttt{An}(v_s)}$$

R_1: Slot value annotation

By R_1, the annotations in Figure 5 entails annotation $(p, v_t) \in \texttt{An}(v_s)$, which is illustrated as the dashed line arrow in the figure. The role of this entailment rule R_1 is to *lift* slot annotations to slot *value* annotations. By using this mechanism, we can annotate media *contents* indicated by slot values.

Now we are ready to annotate contents specified by `url` slot. However, at present, `url` slot of DublinCoreWebble `wm` has no value. To get the url of the image displayed by the image webble `wi`, we can use webble composition and the slot connection mechanism. By combining `wm` with `wi` as a child, and connecting both `url` slots, the `url` slot of `wm` gets a url from `url` slot of `wi`. As a result, the annotation of `wm`'s `url` slot gets the following attribute-value pair.

- `An(wm:#url) = {..., (wb:hasValue, http://a/b/c.jpeg)}`

Then, we apply entailment rule R_1 to associate slot values with annotations of slot values. In this example, rule R_1 matches with $s =$`wi:#url`, $t =$`wm:#url`, $v_s =$ `http://a/b/c.jpeg`, $v_t =$`"La Joconde"`, $p =$`dc:title`. As a result, this example annotation entails `(dc:title, "La Joconde")` $\in$ `An(http://a/b/c.jpeg)`. It is exactly an annotation of the image `http://a/b/c.jpeg`.

As we demonstrated here, our framework uses `wb:hasValue`, `wb:represents`, `wb:annotatesValueOf`, and entailment rule R_1 to associate slot values to annotations. These three terms and one rule form the core of the webble vocabulary. The full definition of the webble annotation core vocabulary can be found in the appendix.

2.3 Mapping Annotations to RDF Graphs

Conceptually a webble annotation is just a set of attribute-value pairs. However, by restricting attribute names to be only RDF properties, we can easily map webble annotations to RDF graphs. The basic idea is that we can represent an attribute-value pair (a, v) in annotation $An(i)$ as an RDF statement "`i a v`" if `a` is an RDF property.

Table 1 shows the mapping from webbles and webble annotations to RDF graphs used in our framework. In our approach, each webble and each of its slots is mapped to an RDF resource. Static structure (ownership of slots, etc.), dynamic state (slot values) and annotations are mapped to RDF statements.

The graph in figure 6 shows an RDF representation of the example that we use in this section. In this graph, statements mapped from webble annotations are

Table 1. Mapping from webbles and webble annotations to RDF graphs

Webbles and annotations	RDF
webble w	`w`
slot s of webble w	`w wb:hasSlot w:#s`
value v of slot s of webble w	`w:#s wb:hasValue v`
pair (a, v) in annotation $An(i)$	`i a v`
webbles and annotations	an RDF graph

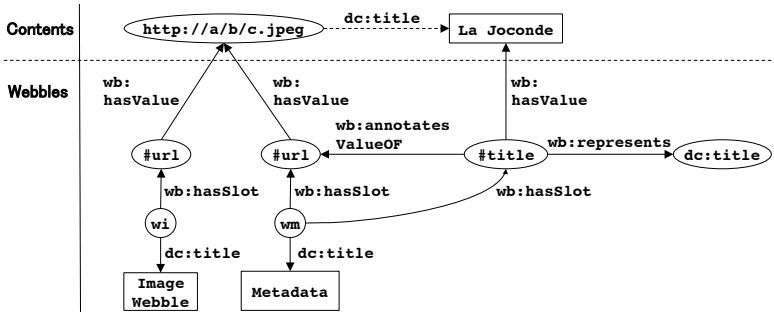

Fig. 6. RDF representation of webble annotation

represented as solid line arrows, and statements inferred by the entailment rule are represented as dashed line arrows. An RDF representation of webble annotations consists of graphs representing annotations of webbles and slots (webble graph), and graphs representing annotations of contents (content graph). As we can see from this example, our framework can produce "clean" content graphs, that is, content metadata is represented by using only standard controlled vocabularies without webble specific vocabularies. This is an advantage for interoperability of webble annotations with existing metadata standards. By storing RDF representations of webbles into webble repositories, we can search webbles by using SPARQL which is a query language for retrieveing information from RDF documents.

2.4 Enhancing Webble Behavior Based on User Annotations

The webble architecture provides a webble composition mechanism and a slot connection mechanism. The design of these mechanisms are intended to combine arbitrary pairs of webbles and slots. As we mentioned in the introduction, such over-flexibility entails the risk of misconnections. Regarding webble composition, the current webble architecture does not provide any information for restricting the combination of webbles to be combined. For slot connection, one possible solution is type-checking of slots. However, basically, webbles are designed and should be designed for preserving interoperability of webbles by choosing their

slot types from a small set of common slot types (string, integer, real, etc.). Additionally, automatic data conversion is implemented between many combinations of common slot types. As a result, selectiveness of slot types is usually too low for helping users to avoid misconnections of slots (static type checking in statically typed programming languages helps us in finding some errors but it is still far from eliminating all errors).

As a consequence, if we want to introduce a mechanism which helps users in reducing miscombination and misconnection of webbles, it has to be application specific. For each application, we need to "plug-in" an additional type system of webbles and slots specific for the application. For this purpose, we can use *user* annotation of webbles. RDF mapping of webble annotations enable us to define custom type systems by using RDF schema (RDFS) or more advanced RDF vocabularies like OWL. Then we can annotate types of webbles and slots used in applications.

By using the example illustrated in figure 2, suppose a user intends to combine the image webble `wi` only with child webbles which describe metadata. In this case, the user can introduce a custom type system of webbles, then declare webbles by using this type system and check whether a webble can combine with another.

Firstly, the custom type system has to be declared as a set of RDF triples.

- `DublinCoreWebble rdfs:subclassOf wb:Webble`
- `DublinCoreWebble rdfs:subclassOf MetadataWebble`
- `CIDOCWebble rdfs:subclassOf MetadataWebble`
- ...

Then, the type of webbles acceptable as children by the image webble `wi` has to be declared as part of `wi`'s annotation. In this example, the acceptable type is declared by using `wb:acceptsAsChild` which is a term of webble vocabulary:

- `An(wi) = {..., (wb:acceptsAsChild, MetadataWebble) }`

Finally, we can check whether a webble can be combined with `wi` or not by evaluating the following rule.

$$\frac{\texttt{x wb:acceptsAsChild t} \wedge \texttt{y rdfs:subclassOf t}}{\texttt{y} \text{ can be combined with } \texttt{x} \text{ as a child}}$$

In the same manner, we can also restrict possible combinations of slots to connect by using annotations and rules. For instance, both image webble and Dublin Core webble have `url` slot whose type is TextSlot. TextSlots can accept texts in general as values. However, `url` slot of image webble has a URL of an image as its value, and `url` slot of Dublin Core webble has a URL of a resource to annotate in general as its value. We can declare such detailed information of slot value domains as annotations.

- `An(wi:#url) = {..., (wb:valueDomain, vra:Image)}`
- `An(wm:#url) = {..., (wb:valueDomain, rdfs:Resource)}`

In the above example, `vra:Image` (which is a class defined by Visual Resource Association [17]) is used to fix the domain of `wi:#url` slot to image URLs. The domain of `wm:#url` slot is also fixed to URLs of resources in general by using `rdfs:Resource`. By using this additional information about slot value domains, we can check whether values of one slot can be set to another slot by using the following rule:

$$\frac{\texttt{x wb:valueDomain s} \wedge \texttt{y wb:valueDomain t} \wedge \texttt{s rdfs:subclassOf t}}{\text{values of slot } \texttt{x} \text{ can be set to slot } \texttt{y}}$$

In this paper, we do not discuss details about webble vocabulary and rules for controlling behavior of webbles. The main point we would like to emphasise here is that by introducing application specific type systems and rules, we can carefully restrict the operations that users can perform. We believe that such restrictions on user's operations for avoiding misconnections of webbles is useful and somehow indispensable when we provide webbles for *casual* users.

3 Conclusion

In this paper we have argued that if the webble technology is to be used in its full strength it should be enhanced in two ways: (a) providing means for searching a potentially huge pool of webbles and (b) controlling webble's behavior by introducing application specific type systems and rules. We have proposed webble annotations as an unobtrusive way to remedy these two deficiencies of the current webble technology. Additionally, by studying an important use case (annotation of digital media contents) we have demonstrated the benefits of using standard, controlled vocabularies in the definition of webble annotations.

We are currently researching several aspects of composite webbles. The main concern is query formulation and evaluation for searching webbles in the webble repository. In this paper, we have demonstrated how webble annotations can be directly mapped to RDF graphs and how our mapping method produces "clean" RDF graphs of content metadata. Therefore we can search media *contents* in the webble repository by using the same manner for querying content metadata. However, the question is how to search media *containers* in the same repository. Webbles may have many slots and sometimes they are composite. Annotations over such complex webbles are represented as complex RDF graphs which is difficult to query. To allow typical users to search complex webbles from the repository, we need to introduce a mechanism for "summarizing" webble annotations into simpler forms to search. We have developed a theoretical framework of metadata summarization in the context of composite document repositories [18]. This framework works with metadata comprising a set of terms from a controlled vocabulary. In future work, we will extend this framework to handle metadata represented as RDF graphs.

References

1. Webble portal, http://cow.meme.hokudai.ac.jp/WebbleWorldPortal/
2. Dublin Core Metadata Initiative: http://dublincore.org/
3. The CIDOC CRM: http://www.cidoc-crm.org/
4. Gill, T.: Building semantic bridges between mquseums, libraries and archives: The CIDOC Conceptual Reference Model. First Monday, vol. 9(5) (2004)
5. Handschuh, S., Staab, S.,, M.: CREAM: creating relational metadata with a component-based, ontology-driven annotation framework. In: Proc. of the 1st International Conference on Knowledge Capture, pp. 76–83 (2001)
6. Knight, C., Gasevic, D., Richards, G.: An ontology-based framework for bridging learning design and learning content. Journal of Educational Technology & Society 9(1), 23–37 (2006)
7. Kitamura, Y., Washio, N., Koji, Y., Sasajima, M., Takafuji, S., Mizoguchi, R. An ontology-based annotation framework for representing the functionality of engineering devices. In: Proc. of Asme IDETC/CIE 2006 (2006)
8. Gene Ontology Consortium.Gene Ontology: tool for the unification of biology. Nature genetics 25(1), 25–29 (2000)
9. Lipscomb, C. E. Medical subject headings (MeSH). Bulletin of the Medical Library Association, 88(3), 265 (2000)
10. Al-Shahrour, F., Daz-Uriarte, R., Dopazo, J.: FatiGO: a web tool for fnding signifcant associations of Gene Ontology terms with groups of genes. Bioinformatics 20(4), 578–580 (2004)
11. Lu, Z.: PubMed and beyond: a survey of web tools for searching biomedical literature. Database: The Journal of Biological Databases and Curation (2011)
12. Bracha, G., Griswold, D.: Typechecking Smalltalk in a Production Environment. In: Proc. of the OOPSLA 1993 Conference on Object-oriented Programming Systems, Languages and Applications, pp. 215–230 (1993)
13. PEP (Python Enhancement Proposals) 3107 Function Annotations: http://www.python.org/dev/peps/pep-3107/
14. Bracha, G.: Optional Types in Dart., http://www.dartlang.org/articles/optional-types/
15. Learning Object Metadata, IEEE Standard 1484.12.1–2000 (2002)
16. The ACM Computing Classification System (2012), http://www.acm.org/about/class/2012
17. Assem, M.V.: RDF/OWL Representation of VRA., http://www.w3.org/2001/sw/BestPractices/MM/vra-conversion.html
18. Sugibuchi, T., Tuan, L.A., Spyratos, N.: Metadata Inference for Description Authoring in a Document Composition Environment. In: Agosti, M., Esposito, F., Ferilli, S., Ferro, N. (eds.) IRCDL 2012. CCIS, vol. 354, pp. 69–80. Springer, Heidelberg (2013)

Webble Core Annotation Vocabulary and Semantics

```
wb:Webble            rdf:type    rdfs:Class
wb:Slot              rdf:type    rdfs:Class

wb:hasSlot           rdf:type    rdfs:Property
wb:hasSlot           rdfs:domain wb:Webble
wb:hasSlot           rdfs:range  wb:Slot
wb:hasValue          rdf:type    rdfs:Property
wb:hasValue          rdfs:domain wb:Slot

wb:represents        rdf:type    rdfs:Property
wb:represents        rdfs:domain wb:Slot
wb:represents        rdfs:range  rdfs:Property
wb:annotates         rdf:type    rdfs:Property
wb:annotates         rdfs:domain wb:Slot
wb:annotates         rdfs:range  rdfs:Resource
wb:annotatesValueOf  rdf:type    rdfs:Property
wb:annotatesValueOf  rdfs:domain wb:Slot
wb:annotatesValueOf  rdfs:range  wb:Slot
```

R_1: Slot value annotation

$$\frac{\begin{array}{c}(\texttt{wb} : \texttt{hasValue}, v_t) \in \texttt{An}(t) \wedge \\ (\texttt{wb} : \texttt{represents}, p) \in \texttt{An}(t) \wedge \\ (\texttt{wb} : \texttt{annotatesValueOf}, s) \in \texttt{An}(t) \wedge \\ (\texttt{wb} : \texttt{hasValue}, v_s) \in \texttt{An}(s)\end{array}}{(p, v_t) \in \texttt{An}(v_s)}$$

R_2: Webble or slot annotation

$$\frac{\begin{array}{c}(\texttt{wb} : \texttt{hasValue}, v_t) \in \texttt{An}(t) \wedge \\ (\texttt{wb} : \texttt{represents}, p) \in \texttt{An}(t) \wedge \\ (\texttt{wb} : \texttt{annotates}, s) \in \texttt{An}(t) \wedge\end{array}}{(p, v_t) \in \texttt{An}(s)}$$