The CellML Metadata 1.0 Specification Research Summary - 18 May 2001

1 Introduction

1.1 Need for Metadata in CellML

Metadata is usually defined as "data about data". It is the supporting information that provides context to a resource. In CellML, the model (i.e., the structure and mathematics of the model) is the resource. Information that puts the model into the larger scientific context is metadata. Metadata in CellML includes information such as the literature reference that supports the model, the identity of the creator(s) of the model, and the species for which the model is relevant.

The CellML metadata project needs metadata for two primary reasons:

- It will be difficult to reuse other people's models and components without metadata to provide the scientific context for these objects. A modeller considering reusing someone else's model component will need to know things such as: what biological entity the component represents, for which species the component is relevant, and when the component was created and last modified (to help determine whether it is likely to incorporate the most recent experimental results).
- As the number of models and components grows, metadata will provide the only scalable method for locating particular models and components. Experience in other biological fields shows that as a field grows, powerful search techniques are needed to enable researchers to find relevant resources. These search techniques require structured metadata.

Metadata in CellML can be used in many different ways, such as:

- To support searches of a model repository (or at least to make it possible to automate loading of a database that will support such searches).
- To enable automatic discovery of models published on remote websites, such as laboratory websites.
- To allow the documentation for a model to be kept in the same document as the model itself, which will keep the documentation from becoming obsolete as work continues on the model.

The metadata structure should be flexible and extensible, because it is almost certain that we have not thought of all possible uses of CellML metadata.

1.2 The Larger Metadata Picture

Metadata has become a bit of a buzzword lately. This is because people are starting to realise that we cannot get the maximum use out of the information stored on the web without metadata. It is currently not particularly easy to find a specific piece of information on the web, and once you have found the information, it is not easy to determine whether or not you should trust it. Metadata can address both of these problems. Therefore, there is a push to begin to incorporate metadata into web resources. Tim Berners-Lee has been particularly active in pushing for a "semantic web", in which resources on the web would include the semantic information necessary to allow machines to *understand* (not just read) them. The W3C has set up a semantic web activity[1]. Some software projects, such as Mozilla[2], have begun trying to take advantage of the metadata that is currently available about web resources.

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

[2] http://www.mozilla.org/

The “semantic web” vision is one of the future, and not of today. Several projects are beginning to take tentative steps towards realising Tim Berners-Lee’s dream, but success is by no means certain. The library science community is leading the way in implementing metadata. A consequence of this is that the tools being provided for handling metadata on the web (such as the Resource Description Framework[3], or RDF) have come from the knowledge management community. Like any academic discipline, that community has its own jargon, which can be a hindrance to the rest of us when we try to understand and use these tools. However, several projects are now using RDF, and a variety of tools have been created for it. These will be discussed in Section 1.4.

None of the problems faced by the nascent metadata community are insurmountable. It seems very likely that something resembling the “semantic web” will come into existence, if for no reason other than the importance of the problem it is attempting to address. Therefore, we should at least consider how we can make metadata in CellML compatible with the semantic web activity.

1.3 Metadata in CellML

The initial step in incorporating metadata into CellML was to determine what sorts of information modellers might want to store about their models, and what sort of information software developers might find useful to be able to store. This was done as part of the requirements gathering for version 1.0 of CellML. A list of the metadata requirements for CellML is included in the requirements document[4].

The necessary metadata can be split into three broad categories:

- Metadata that can be mapped easily onto the Dublin Core elements (see Section 1.5).
- Literature citations, which we might be able to handle using an existing standard such as DocBook[5] or the Object Mangement Group’s Bibliographic Query Service[6].
- Metadata that is specific to biology and/or CellML, an implementation of which we will probably need to develop ourselves.

1.4 RDF

Information about RDF can be found on the W3C’s Semantic Web Activity Page[7].

1.4.1 What is RDF?

There is no doubt about it: RDF is nowhere near as intuitive as XML is. It is pretty clear what the purpose of XML is and how to go about using XML to solve problems. Not so for RDF. However, with a little patience, I think most people can understand what RDF is and what it hopes to accomplish. With a little more work, your average XML geek can figure out how to use and create RDF vocabularies, too.

So what is RDF? Well, the easy answer is that it is the W3C’s recommendation for handling metadata on the web. But that doesn’t tell you much. RDF stands for “Resource Description Framework”, and that really does describe what RDF is. It is a framework that allows you to store descriptions (i.e., metadata) about resources. A resource can be literally *anything*. For the purposes of CellML, resources can be the model document, the model itself, or components in the model.

[3] http://www.w3.org/RDF/
[4] http://www.cellml.org/private/fundamentals/requirements.html
[5] http://www.docbook.org
[6] http://www.omg.org/homepages/lsr/
[7] http://www.w3.org/RDF/

1.4.2 Why Bother With RDF?

RDF by itself does not allow people to store metadata. It merely provides a standard framework onto which various groups can hang their metadata vocabularies. Some benefits of having this standard framework are:

- **It provides a common attribute=value data model for the metadata**. All metadata expressed in RDF can be presented as a series of attributes (i.e., properties of the resource) and their values. For instance, one attribute:value pair for a CellML model might be `species=Mus musculus`.
- **It provides an extensible method for storing metadata of increasing complexity**. Some metadata properties will have simple values, such as the species property shown above. Other metadata properties will have complex values. In the latter case, the value of the metadata property is itself considered a resource, and additional metadata properties are stored about it. This is made more clear by an example. Consider the case of the `model creator` property. This could be given a simple value of the creator's name, such as **`John Doe`**. However, it is more powerful to consider the value of the creator property to be a new resource (the person identified by the name "John Doe"). This allows the person's name to be stored as metadata about the new resource. Why bother doing this? Because it also allows additional metadata to be stored about the person, such as the person's mailing address, phone number, etc. Most importantly, we don't have to know ahead of time what sorts of metadata processing software might want to store about the person. If a particular application wants to store the person's favorite color, it can do so. Other applications might not recognize the meaning of the particular element that stores the favorite color, but they will be able to understand that it is some sort of property about the resource (i.e., person) that is the creator of the model. This allows the application to handle the unknown metadata gracefully (I suspect most applications would at least be able to present the attribute=value pair to the user).
- **It makes it possible for applications that don't know anything about CellML to understand our metadata**. OK, this one isn't a reality yet, but it is part of Tim Berners-Lee's vision of a semantic web. Eventually, search engine tools could become RDF capable. In that case, people would be able to perform much more powerful searches for information on the web. If someone wants to find all web resources created by John Doe, he/she could search explicitly for resources where `creator=John Doe`, instead of just searching for resources that contain the string "John Doe".
- **There are tools out there that use RDF**. It is definitely true that RDF is still a fledgling technology. However, there are tools out there that parse RDF, and tools that actually use RDF to build databases, knowledge stores, and other such things. See the W3C's RDF project list[8] for a list of tools and projects using RDF.
- **RDF provides some good basic data structures**. The RDF team did a good job defining groupings for metadata values, for instance. This makes it easy to allow multiple values for a metadata property, and allows you to unambiguously specify whether the multiple values should be considered an ordered list, a list of alternative values, or just an unorder group. These data structures (called *containers*) can be used with any RDF element, with not additional definition by the RDF vocabulary author.

1.4.3 Multiple Methods for Expressing Metadata in RDF

The plethora of methods for expressing the same basic piece of information is probably Warren's biggest problem with RDF. (Well, maybe the verbosity is his biggest problem, which is saying something, given his penchant for verbose implementations....)

Why does RDF allow multiple methods for storing the same information? I think it has to do with the fundamental database design principle of generalize whenever possible, and overload your classes as

[8] httP://www.w3.org/RDF/#projects

much as possible. The reason this is good database design is that generalizing and overloading are both good techniques to create a flexible data model that will handle new types of information as they become necessary. If you exactly map each type of information to a column in a database table (or an element in an XML schema), new types of information will require new columns (or elements). If you have a generalized structure that handles an entire class of information, you are much more likely to be able to handle any type of information without changing your schema. I suspect that this line of reasoning explains why the basic RDF data model allows some types of information to be stored in more than one way.

Of course the downside is that someone new to RDF has incredible anxiety attacks trying to figure out which way is the "right way", when in fact, all that matters is that he/she pick a method and implement it consistently.

1.5 The Dublin Core

Information about the Dublin Core can be found on the Dublin Core's website[9].

1.5.1 What is the Dublin Core?

The Dublin Core is a group of metadata properties. These properties were identified as "common" across a large range of resources by a group of library science and knowledge management folks. These properties are things like creator, publisher, subject, and date. A full list, with definitions, can be found in the Dublin Core Metadata Element Set, Version 1.1: Reference Description[10].

The Dublin Core Metadata Initiative group has also provided a standard set of "qualifier" elements. These elements add information to the basic elements. Qualifier elements either provide *type* information or *scheme* information. Type information classifies the basic element. For instance, the date element can have a type of created, modified, valid, available, or issued. Scheme information indicates how the content of the element is encoded. For instance, the date element can have a scheme of W3C-DTF or DCMI Period. A full list of qualifiers and their allowed values can be found in the Dublin Core Qualifiers document[11].

It is important to note that Dublin Core does not have to be expressed in RDF. The Dublin Core elements are not elements in the XML sense. They are simply standard names and definitions for common types of metadata. However, the Dublin Core can be expressed in RDF, which is how I recommend the CellML project take advantage of this work.

1.5.2 Why Bother With the Dublin Core?

Not all of the Dublin Core elements are immediately applicable to CellML metadata requirements. So why should we bother with the Dublin Core? Because this set of elements is widely referenced. General purpose tools are more likely to understand the Dublin Core metadata vocabulary than any other vocabulary. Also, it makes it more obvious that certain CellML metadata properties (such as model creator) map directly to metadata properties that are found in other fields. While this may not be of great value to us right now, later on we will probably be glad we took the effort to make our metadata more generally understandable.

1.6 The CellML Metadata Data Model

The basic CellML metadata properties were defined in section 16 of the CellML requirements document.

In the interests of writing the serialization section, and because the data model for CellML metadata hasn't really changed since October of 1999, this section will be done last. Which probably means I won't have time to do it at all!

[9] http://dublincore.org/

[10] http://dublincore.org/documents/dces/

[11] http://dublincore.org/documents/dcmes-qualifiers/

1.7 Specific Serialization Issues

This section assumes that the reader is reasonably familiar with basic RDF constructs such as containers and the `rdf:resource` and `rdf:parseType` attributes. It also assumes a certain familiarity with the Dublin Core element set and qualifiers set.

1.7.1 Use of `<rdf:Description>` elements vs. `rdf:parseType` attributes

Both the `<rdf:Description>` element and the `rdf:parseType` attribute (with a value of `"Resource"`) can be used to create a new resource. When should each be used? Well, this is largely up to the discretion of the vocabulary author. I have used the `rdf:parseType` attribute wherever possible, because it is less verbose. However, the `<rdf:Description>` element must be used if you want to give the new resource an identifier, which would allow the resource to be referenced by other RDF constructs.

1.7.2 Use of the `<rdf:value>` Element

The `<rdf:value>` element is often used when a new resource is created inline, and some metadata is stored about it. The `<rdf:value>` element stores the actual value of the metadata, and additional elements can be used to store qualifying information. If a processing application does not recognize the qualifying elements, it can take the contents of the `<rdf:value>` element and transfer them to the containing element. For instance, in the example shown in Figure 1, an application that only understands the Dublin Core, but not the Dublin Core qualifiers, could reasonably be expected to assign the date metadata a value of 2001-04-24. An application that understands the Dublin Core qualifiers would also know that this is a creation date, encoded using the W3C-DTF format.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.0/"
    xmlns:dcq="http://purl.org/dc/qualifiers/1.0/">

  <rdf:Description about="#cellml_element_id">
    <dc:date rdf:parseType="Resource">
      <dcq:dateType>created</dcq:dateType>
      <dcq:dateScheme>W3C-DTF</dcq:dateScheme>
      <rdf:value>2001-04-24</rdf:value>
    </dc:date>
  </rdf:Description>
</rdf:RDF>
```

FIGURE 1: Date metadata that uses the `<rdf:value>` element.

1.7.3 Use of Qualifiers

The Dublin Core qualifier scheme is such a good one that I copied it extensively in other areas of the CellML metadata. Any time I wanted to provide a classification for a piece of metadata, I used a *type* qualifier. Any time I wanted to provide information about how the metadata information is encoded, I used a *scheme* qualifier. These elements are named by taking the name of the containing element and appending "`_type`" or "`_scheme`".

1.7.4 Controlling the Value of a Metadata Property

It is often desirable to control the value of a metadata property. This is particularly true for qualifiers. For instance, knowing that the date is encoded using "W3C-DTF" doesn't do you much good if you don't know what "W3C-DTF" means. There are several ways to control the value of a metadata property in RDF. I've listed them below, with an explanation for when I think each method should be used:

- **Use the `rdf:resource` attribute**. The `rdf:resource` attribute can be defined on an RDF element to indicate that the value of the element is a resource, which is identified by the value of the `rdf:resource` attribute. I recommend using this method to control the value of a metadata property when the value subtypes the containing resource, and the subtyping adds properties to the metadata content of the containing resource.

  "What did she just say?" I hear you cry. Let me try to explain. The *containing resource* is the RDF element that creates a resource to represent the thing about which we are trying to define metadata. The best example of this from the CellML metadata is the definition of a literature citation for a CellML model. The citation is represented by the `<bqs:reference>` element. This creates a resource to represent the journal article, or whatever it is we are trying to reference, and contains a lot of metadata about the article. One of the pieces of metadata is the `<bqs:reference_type>` element. This element identifies the subtype of reference: is it a journal article, book article, web resource, or what? Additional metadata properties are required for a reference based upon which subtype is identified. If it is a journal article, we need to identify the journal, the volume, and the first and last pages. If it is a book, we need to identify the publisher, edition, volume, etc. The data structure for each of these subtypes is represented in the RDF schema. That is, there is a resource (in the RDF schema) that defines the additional metadata content added to the reference resource by the subtyping. I think the best way to serialize this is to make the URI that identifies the resource (in the RDF schema) that defines the new metadata properties on the containing resource.

- **Define a controlled vocabulary and enforce it in the RDF schema.** This method can be used whenever you think you know ahead of time all of the allowed values for the metadata. The Dublin Core qualifiers are good examples of this method. The Dublin Core folks have defined certain qualifiers for their elements, and defined certain allowed values for these qualifiers. An RDF schema implementing the Dublin Core could limit the values of the qualifier elements to the values defined in the DC qualifiers document. The value of the qualifier element adds information to the metadata about the containing resource, but does not add content to the metadata allowed on the containing resource.

- **Define a standard method for storing the information, and recommend some values**. If you think it is likely that you don't know all the values that an element may receive, but would still like to recommend some limitations, you can define the meaning of certain values for the element in the textual documentation on your scheme, and/or in any technical specification you provide for your RDF vocabulary. A good example of this is the `<cmeta:species>` element. The CellML metadata specification will recommend that the value of this element be a scientific name for a species, but the accompanying RDF schema will not enforce this.

1.7.5 Use of the `rdf:resource` Attribute

An `rdf:resource` attribute is used to indicate that the value of the metadata property represented by the element is a resource, which is identified by the value of the `rdf:resource` attribute. I use this attribute in two ways:

1. As a way to control the value of a qualifier element that is subtyping another element, as described in Section 1.7.4.

2. When the value of an element is a resource that is defined elsewhere. For instance, in the BQS serialization, information can be defined about a journal, and then referenced via an `rdf:resource` attribute in a journal article citation.

Technically, the first case is just a special instance of the second case. The resource identified by the value of the `rdf:resource` attribute is in the RDF schema, but it is still a resource.

1.8 Use of RDF Containers

RDF provides three types of containers for multivalued metadata: bags, sequences, and alternatives. The RDF specification defines the meaning of each of these. A bag is used if a metadata element has unordered, multivalued content. An example is the students in a class. A sequence is used if a metadata element has ordered, multivalued content. An example is the authors on a scientific paper. An alternative is used if a metadata element has single-valued content that is chosen from a list of alternatives. An example is the title of a document provided in multiple languages.

Warren is concerned that not every type of RDF container may be relevant for each metadata element in the CellML metadata specification, and wonders if we need to restrict the use of containers. This should not be necessary. The RDF specification already defines the meaning of each type of container. The recommendations in the CellML metadata specification will demonstrate how to use containers to represent the types of metadata that CellML metadata compliant software must recognize. If software chooses to become CellML metadata compliant by implementing only the types of metadata defined in the CellML metadata specification, it need only recognize the containers shown in the recommended definitions in the specification. If software choses to become CellML metadata compliant by becoming fully RDF capable, the RDF specification will define the meaning of the different containers for us.

1.9 Use of Controlled Vocabularies of Terms

There are several possible methods for dealing with controlled vocabularies of terms in RDF metadata:

- List the terms and their meanings in the CellML metadata specification, and leave it to processing software to restrict the allowed values of the relevant metadata elements.
- Use the `<rdfs:range>` property in the RDF schema to limit the allowed values for an element to a specified set. (The terms and their meanings will also be listed in the CellML metadata specification).
- Use the `rdf:resource` attribute on the metadata element to point to a namespace URL.

The first two options produce the same RDF in the CellML document, an example of which is shown in Figure 2. This is because the control of the vocabulary is handled in a separate document (the CellML metadata specification for option 1 and the CellML RDF Schema for option 2). The second option is preferable to the first because it produces machine-understandable limits on the allowed values of the element. Since these allowed values would also be defined in the specification, the second option does not produce any additional demands on CellML processing software: software could still ignore the RDF Schema and implement only the recommendations in the CellML metadata specification.

The third option produces the RDF shown in Figure 3. The metadata content in the two figures is identical.

One important difference between the options presented in the two figures is the method by which modellers can use their own terms as values for the RDF element. If the terms in the vocabulary are controlled by the RDF schema and the CellML metadata specification (Figure 2), modellers would need to place the metadata element in their own namespace in order to use a term not in the CellML controlled

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:cmeta="http://www.cellml.org/metadata/1.0#">

  <rdf:Description about="some_element_id">
    <cmeta:math_problem rdf:parseType="Resource">
      <cmeta:math_problem_scheme>GAMS</cmeta:math_problem_scheme>
      <rdf:value>I1a</rdf:value>
    </cmeta:math_problem>
  </rdf:Description>
</rdf:RDF>
```

FIGURE 2: An example of RDF metadata produced by controlling vocabularies of terms in the specification and the RDF schema.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:cmeta="http://www.cellml.org/metadata/1.0#">

  <rdf:Description about="some_element_id">
    <cmeta:math_problem rdf:parseType="Resource">
      <cmeta:math_problem_scheme
          rdf:resource="http://www.cellml.org/metadata/terms#GAMS/" />
      <rdf:value>I1a</rdf:value>
    </cmeta:math_problem>
  </rdf:Description>
</rdf:RDF>
```

FIGURE 3: An example of RDF metadata produced by controlling vocabularies of terms using the **`rdf:resource`** attribute.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:cmeta="http://www.cellml.org/metadata/1.0#"
    xmlns:app="http://www.bozo.com/my_app/">

  <rdf:Description about="some_element_id">
    <cmeta:math_problem rdf:parseType="Resource">
      <app:problem_scheme>my classification scheme</app:problem_scheme>
      <rdf:value>system of ODEs</rdf:value>
    </cmeta:math_problem>
  </rdf:Description>
</rdf:RDF>
```

FIGURE 4: An example of using a term not in the CellML controlled vocabulary if the controlling vocabularies of terms are defined in the specification and the RDF schema.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:cmeta="http://www.cellml.org/metadata/1.0#">

  <rdf:Description about="some_element_id">
    <cmeta:math_problem rdf:parseType="Resource">
      <cmeta:math_problem_scheme
          rdf:resource="http://www.bozo.com/my_app/terms#sillymath/" />
      <rdf:value>ODE</rdf:value>
    </cmeta:math_problem>
  </rdf:Description>
</rdf:RDF>
```

FIGURE 5: An example of using a term not in the CellML controlled vocabulary if the controlling vocabularies of terms are defined using the `rdf:resource` attribute.

vocabulary, as shown in Figure 4. If the terms in the vocabulary are controlled using the `rdf:resource` attribute, modellers would need to supply their own URL as a value for this attribute, as shown in Figure 5.

Recommendation: Control vocabularies in the RDF schema, and include an explanation of all terms in the CellML metadata specification (this results in RDF as shown in Figure 2 and Figure 4). This method is marginally preferable to the method shown in Figure 3 and Figure 5, because that method mixes RDF abbreviated syntax with the full syntax. Mixing the two syntaxes is legal in RDF, but using such a mixed syntax would increase the demands on CellML processing software.