HAL
archives-ouvertes.fr

Formalization of the neuro-biological models for spike neurons.

Lucas Ferro, Charf-Eddin M'Sakni, Pierre Repetto, Nader Salman, Manh-Tien Nguyen, Thierry Viéville

▶ **To cite this version:**

Lucas Ferro, Charf-Eddin M'Sakni, Pierre Repetto, Nader Salman, Manh-Tien Nguyen, et al.. Formalization of the neuro-biological models for spike neurons.. [Technical Report] RT-0324, INRIA. 2006, pp.110. <inria-00080566v2>

HAL Id: inria-00080566
https://hal.inria.fr/inria-00080566v2

Submitted on 24 Jul 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Formalization of the neuro-biological models for spike neurons.

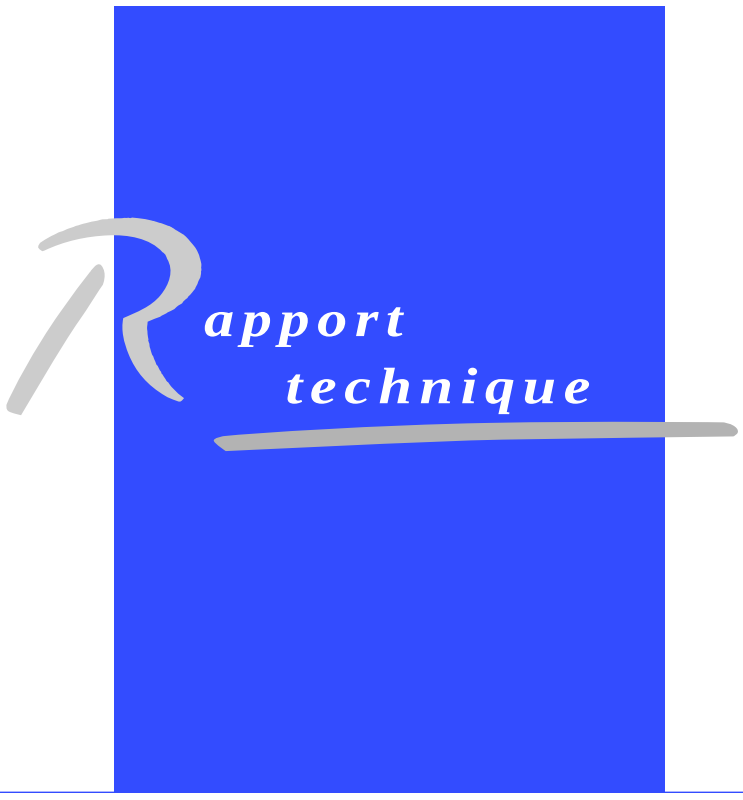
Luca FERRO — Charf-Eddin M'SAKNI — Pierre REPETTO — Nader SALMAN —

Thierry VIÉVILLE — Manh-Tien NGUYEN

N° 0324

June 2006

Thème BIO



ISSN 0249-0803

Formalization of the neuro-biological models for spike neurons.

Luca FERRO , Charf-Eddin M'SAKNI , Pierre REPETTO , Nader SALMAN , Thierry VIÉVILLE * , Manh-Tien NGUYEN †

Thème BIO — Systèmes biologiques
Projet Odyssée

Rapport technique n° 0324 — June 2006 — ?? pages

Abstract:

When modelizing cortical neuronal maps (here spiking neuronal networks) within the scope of the FACETS project, researchers in neuro-science and computer-science use NeuroML, a XML language, to specify biological neuronal networks. These networks could be simulated either using analogue or event-based techniques. Specifications include :

- parametric model specification
- model equation symbolic definition
- formalization of related semantic aspects (paradigms, ..)

and they are used by "non-computer-scientists". In this context XML is used to specify data structures, not documents.

The first version of NeuroML uses Java to map XML biological data which can be later simulated within GENESIS, NEURON, etc. The second version uses tools for handling XML data, as XSL, to transform an XML file.

To allow NeuroML to be used intensively within the scope of the FACETS project, we will entirely analyse the software. First we are going to evaluate this software deeply in the Technical Report section. Then we will propose a prototype to write down NeuroML code easily.

Key-words: NeuroML, Biological Neurons, Spiking Neurons, Neural Networks Simulation.

* Advisor
† Co-worker

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Formalisation des modèles neuro-biologiques à spikes.

Résumé :

Lors de la modélisation de cartes neuronales (réseaux de neurones événementiels) du cerveau réalisée dans le cadre du projet Européen FACETS, les chercheurs en biologie et en sciences de l'information utilisent NeuroML. Il s'agit d'un langage XML permettant de spécifier les modèles de réseaux de neurones biologiques. Ces réseaux sont ensuite simulés de manière analogique ou événementielle. Plus précisément ils contiennent :

- les paramètres de ces modèles
- les équations qui régissent ces modèles
- les éléments qui décrivent la sémantique de ces modèles

et ils sont spécifiés très souvent par des "non-informaticiens". Dans ce contexte XML est donc utilisé pour spécifier non pas des documents mais des données structurées.

La première version de NeuroML utilise Java comme outil de transformation/mapping des données biologiques vers des données pour les simulateurs (GENESIS, NEURON, etc.). La deuxième version utilise des outils de manipulations des données XML, comme XSL, pour réaliser cette transformation.

NeuroML nécessite donc une mise à plat avant son utilisation intensive dans le cadre du projet FACETS. Nous commencerons donc par évaluer cet outil d'une manière approfondie dans la section Technical Report. Puis nous proposerons un prototype permettant d'utiliser NeuroML de façon commode et simple.

Mots-clés : NeuroML, Neurones biologiques, Neurones à spikes, Simulation de réseaux de neurones.

Technical Report

1 NeuroML v1.0: Java

1.1 Using Java

NeuroML software engineers use the Object-oriented style of representation because it seems to be useful in a variety of reasons. First, Java is great for its Reflection as well as its good runtime type information that makes exposing parts of a program to a scripting engine easy. The Java language is extremely portable due to its virtual machine that controls the programming environment. Consequently there is no chance of corrupting the memory. Thus, when a bug happens, there is more evidence about the cause, than in a native written program. A second reason is, that it is extremely simple to code in Java because of IDE tools such as *Eclipse*, *JBuilder*... The final reason for the utility of Java is, that it comes with an extremely large standard library. The library proposes packages for **Schemas**, **XML**, **JAXP**, **JDOM** etc... that are interesting in this project context.

1.2 NeuroML Architecture

NeuroML is specifically geared towards neuron based computational models. The language elements are not merely descriptive, but are defined with careful attention to how a simulation environment can use them to produce executable models. NDK[1] engineers used a simplified form of Java Classes to define the schema and vocabulary of NeuroML.

This Java-based development kit uses a technique called data binding for automating the process of generating and parsing XML from objects in memory. This NDK uses a restricted subset of Java as well as Javas reflection. This includes :

- simple types: int, double, string
- collections, references
- classes and inheritance

The representational framework that is used is based on object-oriented programming concepts. One of the main ideas used is the concept of inheritance applied to templates. All templates are derived from a specific one called neuroml, illustrated in [[Figure 1.1]]. The neuroml template, root of the template hierarchy, itself contains few attributes; it is normally not used to represent objects in and of itself. Templates derived from neuroml add gradually more information. The definition of the templates and our related analysis are presented beginning of **section 1.1.3**.

[1] NeuroML Development Kit

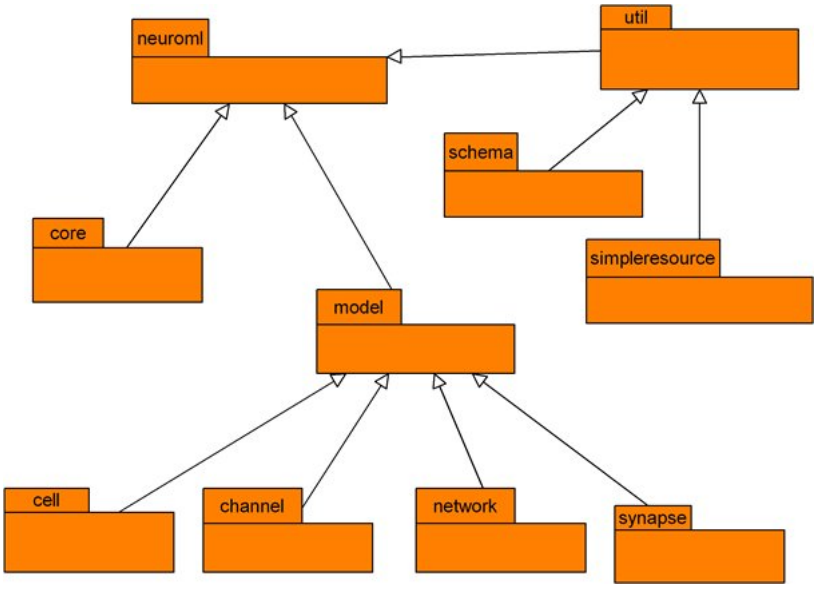


Figure 1: All templates are derived from neuroml. Open arrows indicate inheritance, pointing from inheritors to their parents.

1.3 NeuroML Development Kit analysis

The NeuroML Developpment kit **(NDK)** is composed of 3 complementary packages [[Figure 1.1]]:

neuroml.util this package contains all the tools necessary to manage the files of *jar* type. Moreover, one finds the tools to analyze, create and to read XML documents. Finally, one finds classes which manage the beans objects.

neuroml.core this package contains the various types which constitute the neurons. One finds for example DValue, DoubleArray, StringArray, ...

neuroml.model this package contains the various components of a neuron: the cell (cell), the channel (channel), and the networks (network) of neurons.

Let us now detail positive and negative points of this Development Kit.

1.3.1 Positive points

The project as it has been structured highlights the role of each class. I.e. , the role of each class is highlighted by the name of its package and its own name, illustrated in [[Figure 1.2]]. The classes which manage the *jar* files are really well made, with professional programming,

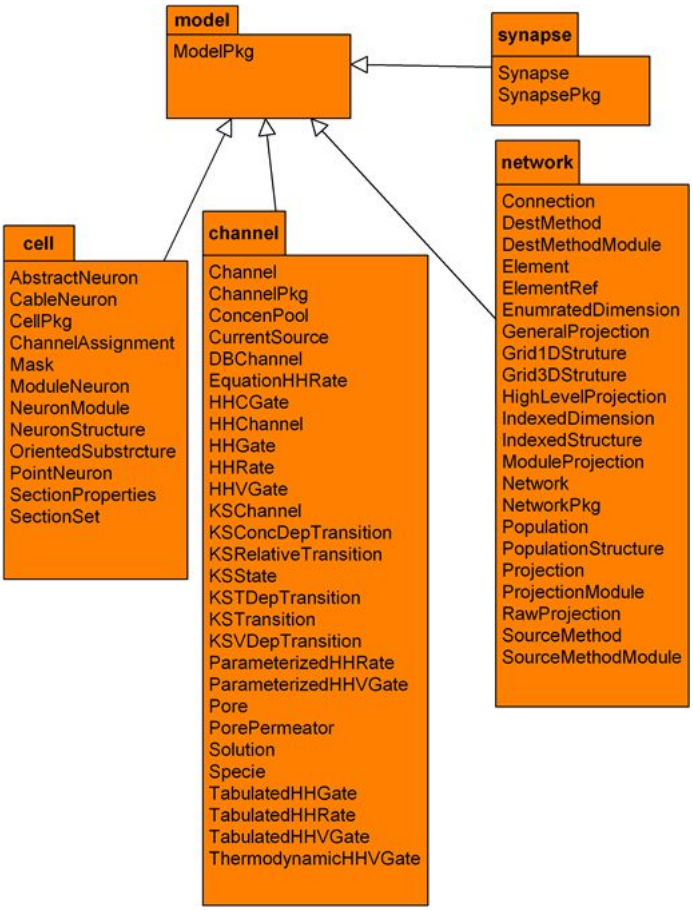


Figure 2: Each package represents its biological counterpart. Open arrows indicate inheritance, pointing from inheritors to their parents.

as is illustrated below.

```
/* Add a manifest file at current position in a stream */
public void stream(OutputStream os, Vector extraFiles) throws
IOException {

  /* the first header in the file should be the global one.
  * It should say "Manifest-Version: x.x"; barf if not */
  MessageHeader globals = (MessageHeader) entries.elementAt(0);
  if(globals.findValue("Manifest-Version") == null)
```

```
  {
    throw new IOException("Manifest file requires " +
    "Manifest-Version: 1.0 in 1st header");
  }
  PrintWriter ps = new PrintWriter(os); globals.print(ps);
  for (int i = 1; i < entries.size(); ++i) {
    MessageHeader mh = (MessageHeader) entries.elementAt(i);
    mh.print(ps);
    /* REMIND: could be adding files twice!!! */
    String name = mh.findValue("name");
    if (extraFiles != null && name != null) {
      extraFiles.addElement(name);
    }
  }
}
```

1.3.2 Negative points

The Java platform proposes, according to an anlyse, more adapted solutions in certain cases than the ones chosen by NeuroML engineers. Here are the things that impairs the Java design of NeuroML :

- The bad use of the attributes access level in the majority of the classes.
  Here is a short example taken out of the *CellPkg* class of the *neuroml.cell* package

  ```
  public Set solutions = new Set("Solution");
  public Set sectionProperties = new Set("SectionProperties");
  public Set neurons = new Set("AbstractNeuron");
  public Set structures = new Set("NeuronStructure");   (*)
  ```

  Here is another example taken from the *Mask* class of the *neuroml* package

  ```
  public String functionName = "All";
  public String functionExpr = "1.0";
  ```

- Creation of a List type which does not have any direction because it inherits of Set and does not bring anything more compared to the *Set* class even in its use.
  Here is the java code of the List class:

  ```
  public class List extends Set {
    public List() { super(); }
    public List(String type) { super(type); }
  }
  ```

- Some attributes were initialized twice: Once at the time of their declaration and then those same attributes are modified in the constructors.
  Here is an example illustrating this phenomenon taken out of the demo.basic.ExampleCell:

  ```
  CellPkg cpkg = new CellPkg("ExampleCells");                (**)
  ```

  The first initialization has been done in (*) the second one is done as shown in the constructor (**)

- Certain parts of the code can be factorized by using the polymorphism of Java, as illustrated in [[Figure 1.3]].

- We consider that using a class such as XMLOut to display the informations of the different Neurons is BAD the solution will be illustrated in the section 4.
  Here is an example of the error we've talked about:

```
if (ob instanceof Reference) {
  elemAppendNVAttr( indentLevel, header, "Reference",
  attribs,((Reference)ob).getTargetName() );
  onlyBasic = true;
} else if (ob instanceof java.util.Collection) {
  // Whoops, printing out a top level collection...
  // This only happens if this is in a set of sets
  // See field.write below Collection
  set = (Collection)ob;
  int subtab = 2;
  // Convert set name...
  String sn = nameOfSet;
  String stag = setPrefix+(docaps ? capitalize(sn) : sn);
  subtab = 1;
  String elType = "java.lang.Object";
  if (set instanceof ContentRestricted) {
    elType = ((ContentRestricted)set).getContentClassName();
  }
  Iterator it = set.iterator();
  while (it.hasNext()) {
    Object sub = it.next();
    writeObject(sub, contents, indentLevel+subtab,
                elType, sn, null);
  }
}
```

- We have to use the JDOM package of Java as mentioned in the NDK to write XML documents.
  Here is an example of the error we've talked about:

```
toptag = "<"+chopClass(top.getClass().getName());
bottag ="</"+chopClass(top.getClass().getName()+">");
```

1.4 A Solution proposal

As listed in the section 1.1.3, there are some dirty things to be revised, in the NeuroML Java Development Kit **(NDK)**. In this section we give the solutions that we've chosen to overcome these problems.
One of the most recurrent points is the bad use of the attributes access level, in Java it is almost never a good idea to make member variables **public**. First, the **public** access level is used for those methods that are necessary to use an object, but if other objects need to access to attributes, we have to create getters and declare them eventually as final!! A second reason is that, information hiding and Javas encapsulation are always good things, even between super and sub classes. A final reason for the non-utility of full public attributes is that, if the software engineers wish to change their interface to use getter/setter functions, they will have to modify all callers of their interfaces, which is onerous at best and in many cases, impossible!! (Especially because NeuroMLs code is extremely dense and is used by other people: as us).
Here is a short solution proposal: The private attributes of an object are initialized in the default constructor as illustrated in [[Figure 1.4]]

We've also noticed that the `intArray`, `StringArray`, `DoubleArray`, had approximately the same structure. However, Java notably offers a way to share (to factor) common features (attributes and methods) between classes, leading to hierarchies without multiple declarations of the same feature: Inheritance. In our approach we propose to use inheritance relationship between those classes to improve the NeuroMLs reusability and to address small evolution and refactoring in it. Following this idea, we created a super class that we've called Array, all of the 3 classes are derived from Array [[Figure 1.5]].
Here is the simple code to do what we've talked about: We've also noticed redundancy in the code of Set.java and List.java classes of the *neuroml.core package*. The second class can be eliminate for 2 reasons: first, `List` paraphrases the Set class in all of its contents; Second reason is that `List Objects` are created and used in all of the packages exactly the same way as `Set Objects`.
We've also noticed that the XML documents were not produced using the JDOM package of Java. The JDOM package is useful for a variety of reasons. First, some characters such as $<$ or $<=$ are not well analysed by the XML processor, however using JDOM allows to step side this problem. A second reason is that, by choosing the JDOM package, the number of code lines is extremely reduced (more or less from 650 to 300 lines). A final reason for the utility of using the JDOM package is that the NDK software can be made modular and extensible. Following this idea, we've introduced a `toStringXML` method in several classes, the `XMLOut`

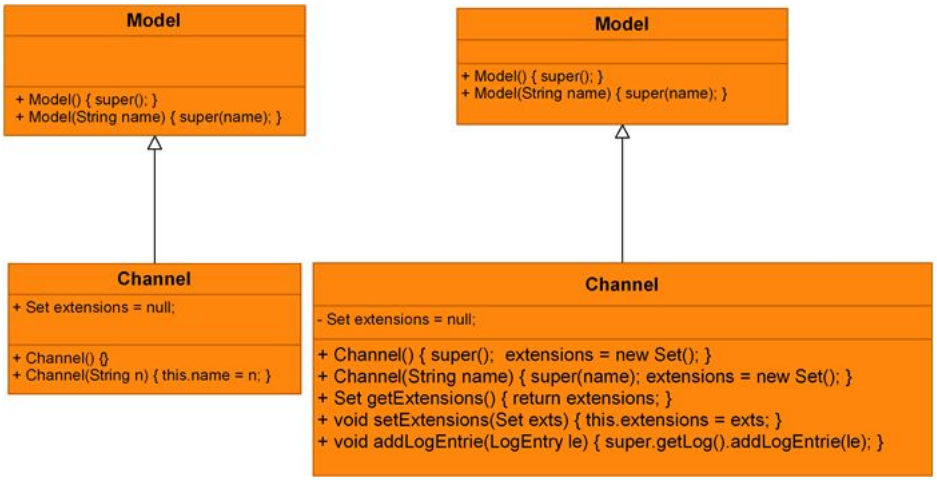


Figure 3: Here is how we change a public attribute to a private one. Open arrows indicate inheritance, pointing from inheritors to their parents.

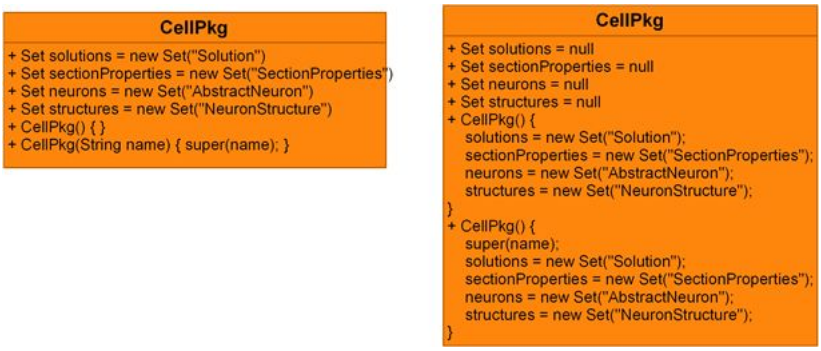


Figure 4: Initialization of attributes in the default constructors. Open arrows indicate inheritance, pointing from inheritors to their parents.

file code has been modified to call these `toStringXML` methods. This new architecture is illustrated in [[figure 1.6]].

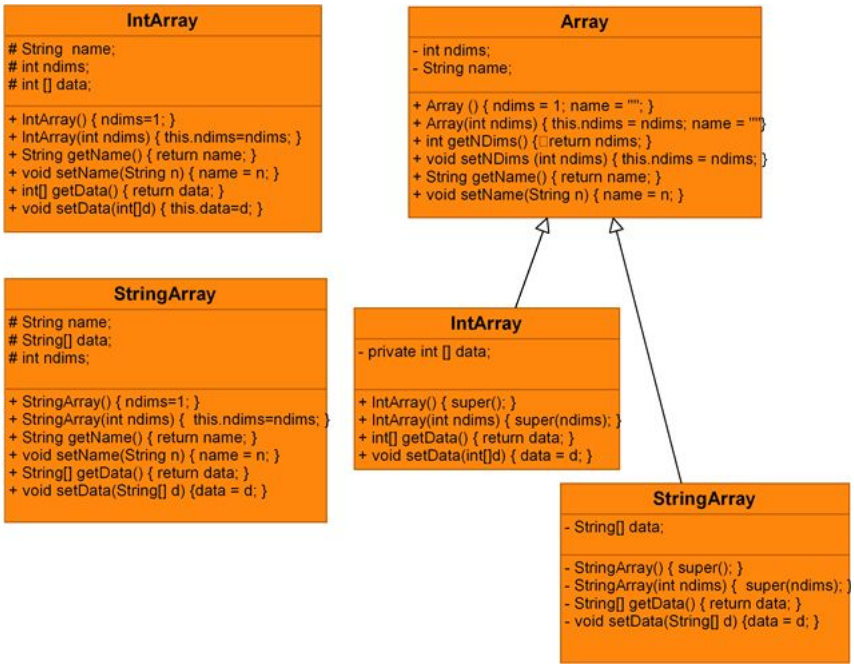


Figure 5: Factorizing StringArray and IntArray code. Open arrows indicate inheritance, pointing from inheritors to their parents.

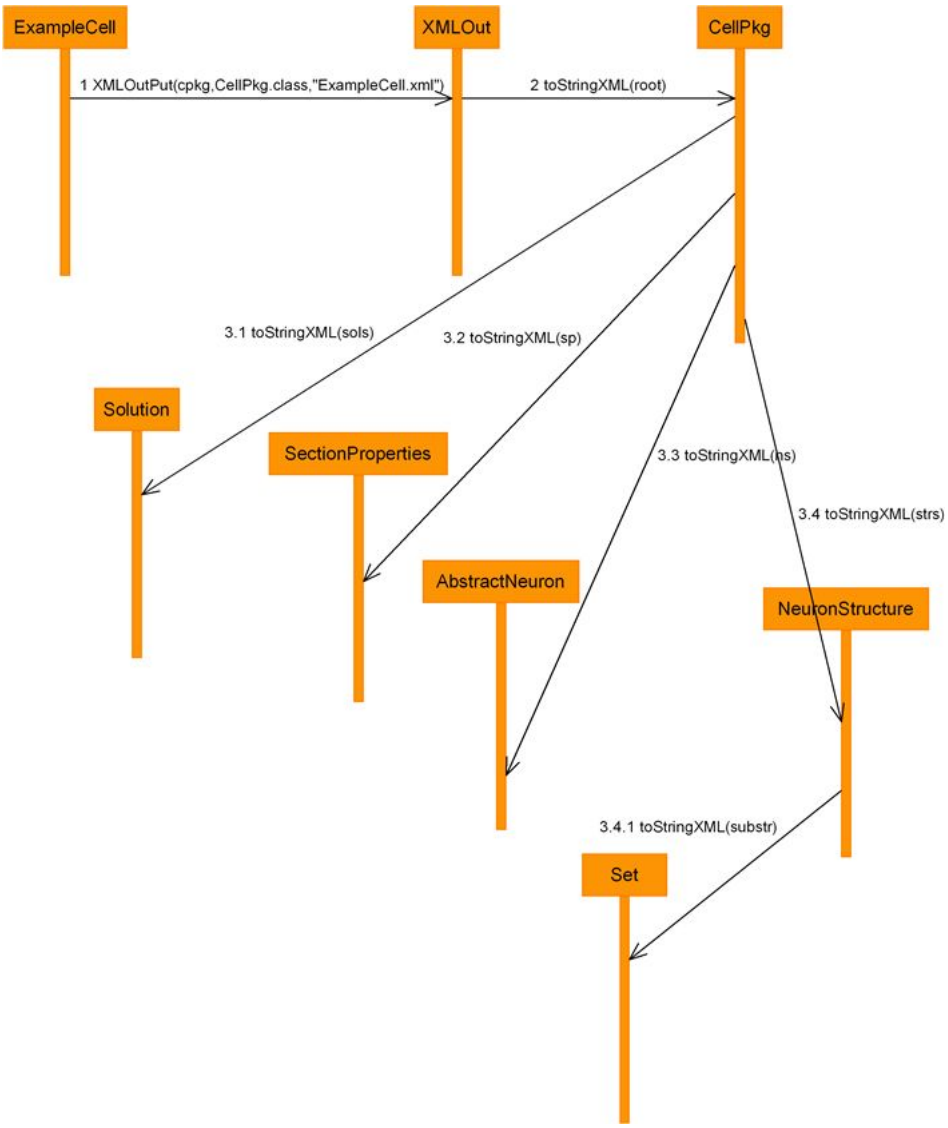


Figure 6: Activity diagram illustrating the use of the toStringXML method.

2 NeuroML v1.1: XML

Presentation

1.0 and 1.1 is the disappearance of the java support. Indeed, all is written in XML. NeuroML 1.1 is now a gather of three main languages: MorphML, ChannelML and Biophysics. It is a web validator operating as a hub. It provides validation tools and transformations tools.

2.1 Validation

An outline of different validation tools
XML probably is one of the main data-exchange formats of Internet. Because of its variety and complexity, data must often be validated before being exploited. A large number of validation tools exist, each with its own grammar and peculiar features. Since there are different aspects which may be in need of a check (such as data structure, format constraints, datatyping, etc.), each validation tool may be more appropriated for a specific task (or it could simply make it easier).
In this document only a part of the validation appliance ensemble will be studied as a possible alternative to XML Schemas for NeuroML specifications.

XML DTD (Document Type Definition)
It is the standard XML schema language of the "past". If compared to other validation tools, it shows limited capabilities. DTD possesses a non-XML syntax and a Document Type Definition Sheet typically consist of a set of declaration blocks that describe a class or a type (datatyping is very limited).

XML Schemas
XML Schema is much more expressive than DTD and it's well recognized by a wide variety of applications. This object-oriented schema language is capable of defining structure, content and semantics of XML documents and possesses many useful features, like inheritance (for elements and attributes) or user-datatype definition. It was published as a W3C Recommendation in May 2001.

RELAX NG (REgular LAnguage for XML Next Generation)
RELAX NG is an easy-to-learn schema language created by unifying Murata Makoto's RELAX Core and James Clark's TREX (Tree Regular Expressions for XML). It possesses both an XML syntax and a compact non-XML syntax. This language can specify patterns for the structure and content of an XML document in a relatively simple but powerful way. Recursion and non-determinism are allowed. RELAX NG is specified by OASIS and also by part two of the international standard ISO/IEC 19757.
Since RELAX and TREX have merged to create RELAX NG, all future development will take place as part of the RELAX NG effort.

Schematron
Schematron is a rules-based schema language capable of finding tree patterns in the parsed document (thanks to XPath expressions). It differs from many other schema languages mainly designed to express the structure of a model and can be used in conjunction with DTDs, RELAX NG or XML Schema, helping in individual rules that are difficult or impossible to express with the partner language.

Abstract comparison
Before getting into a specific analysis between NeuroML needed features which may be offered by RELAX NG and Schematron, we'll compare the general traits of three interesting schema dialects. DTD will only be resolved as a reference, since we consider it as a "surpassed" (but still well spread and useful) tool.

Feature	XML DTD	(W3C) XML Schemas	RELAX NG	Schematron
General overview	an XML structure definition with a list a legal elements	an object-oriented XML schema language	a schema language created by unifying RELAX Core and TREX	a rules-based XML schema language
Grammar	posses its own compact but non-XML grammar	object-like, XML syntax	both an XML syntax and a compact non-XML syntax	XML syntax
Datatyping	no (weak, only applies on attributes)	yes	plugged from W3C XML Schema and others	not directly (can be implemented with user-made rules)
Support for XML namespaces	none	yes	yes	yes
Can directly partner with other schema languages	no	no	partially (with a separate datatyping language)	yes (can be embedded inside XML Schema or RELAX NG)
Post-Schema-Validation Infoset	yes	yes	no	not directly
Complexity	intermediate	quite complex (definitions require considerable expertise)	relatively simple	easy to learn (only six basic elements) but needs XPath knowledge (and XSLT)

Feature	XML DTD	(W3C) XML Schemas	RELAX NG	Schematron
Can express non-determinism	no	no	yes	n.c.
Rules expression	no	no (can only use regular expressions to constraint data values)	no (can only use regular expressions to constraint data values)	yes, using XPath
Data structure description	yes	yes	yes	only using user-made check rules (not direclty)
Data integrity (identifiers, references)	yes	yes	using features of an external datatype system (as W3C XML Schema)	only using user-made check rules (not directly)
Overall flexibility	poor	good (but weak support for unordered content)	high for structures	top, but all must be defined by user
Notes	even if widespread, it is probably going to disappear because of newer and more powerful schema languages	an XML Schema is relatively easy to extend and good for data-oriented application	it relies on both strong mathematical theory underlying regular expression and solid theoretical basis	it is based on a simple idea : finding nodes and checking properties on them

XML Schemas is the current validating tool for NeuroML. It certainly offers many optimal solutions and still be one of the most valuable mechanism to describe composite and intricate structures. Before proceeding, the current NeuroML Schemata specifications (v1.1) will be analysed.

NeuroML specifications according to the XML Schemas proposed
XSD file for NeuroML_v1.1 delegates the definition of each biological "sub-structure" to MorphML_v1.1, ChannelML_v1.1 and Biophysics_v1.1 . It also adds meta-informations about structures, using the Metadata_v1.1 XML Schema.

Thus, a NeuroML element always posses a group of metadata (notes, processing directives, etc.), a group of metadata about references (publications, references, etc.) and a collection of cells (at least one cell is needed). A cell structure is defined by MorphML and it is enriched by biophysical properties described thanks to Biophysics.

It can, but not necessarily, contain a single collection of channels. Each element which follows may be present inside the collection : some definitions of ions involved, a definition of a voltage/concentration dependent cell membrane conductance, a definition of a synaptic process, a concentration of a particular ion species. As for cell structures, each element of a channels collection is defined by ChannelML.

Also, it may have a collection of metadata for each property (a property is a tag/value/type tuple) and a collection of metadata for each group (a group allows elements to be associated, such as for grouping segments or cables into the basal arbor).

An important remark : an XML document being validated (for example) by MorphML_v1.1.xsd will also be accepted by NeuroML_v1.1.xsd, but it won't be validated against ChannelML specific Schemata.

The following tree-structures describe each element of the whole NeuroML validating apparatus. For more details, please refer to NeuroML_v1.1 Schemata sourcecode.

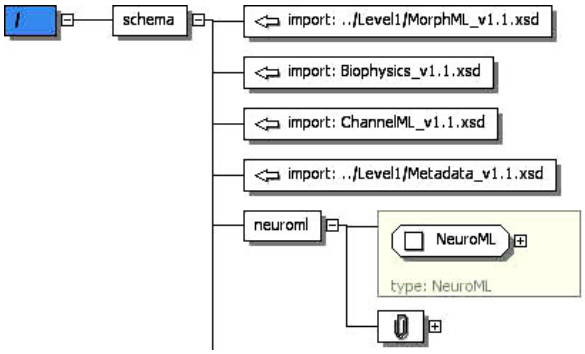


Figure 7: NeuroML v1.1 part A

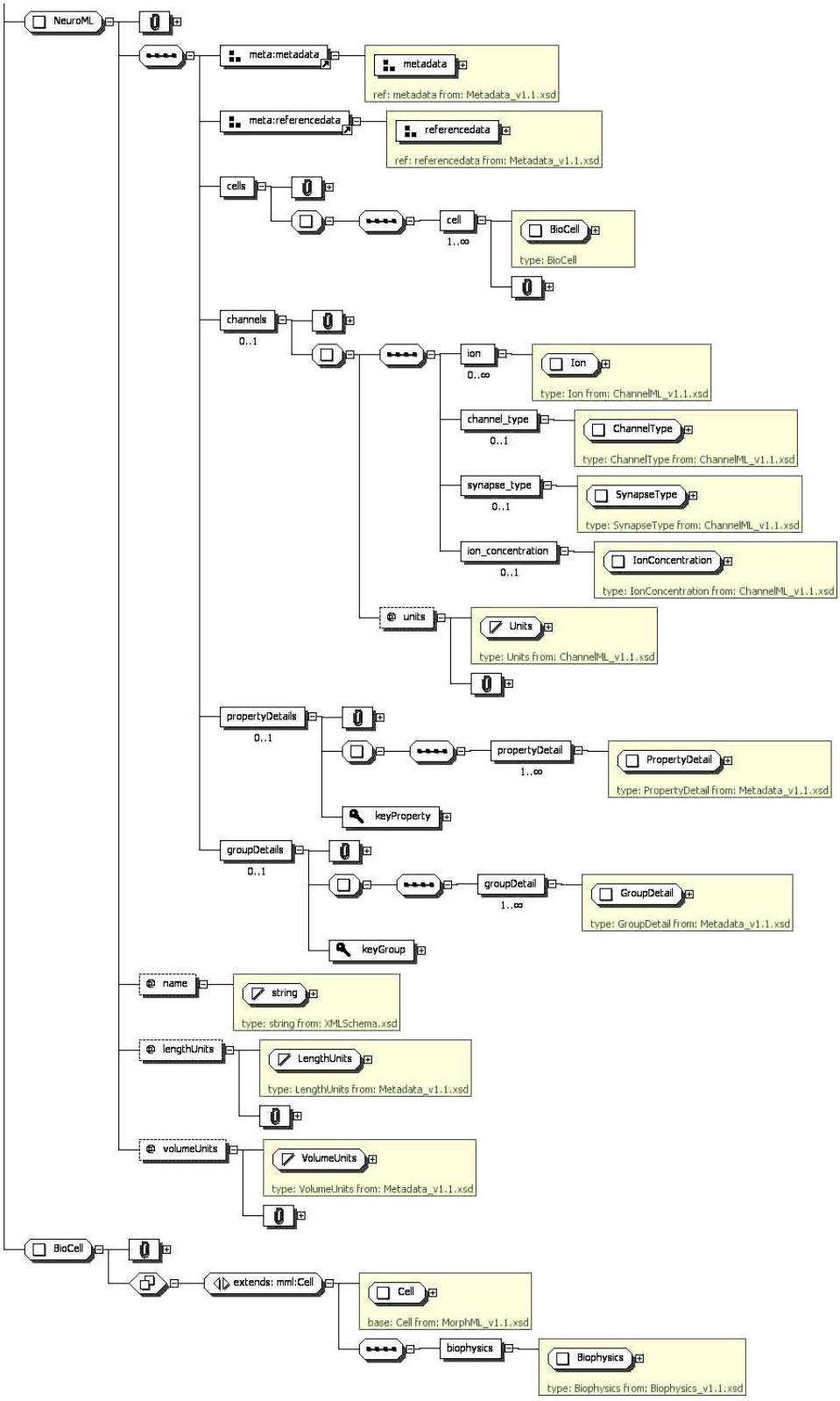


Figure 8: NeuroML v1.1 part B

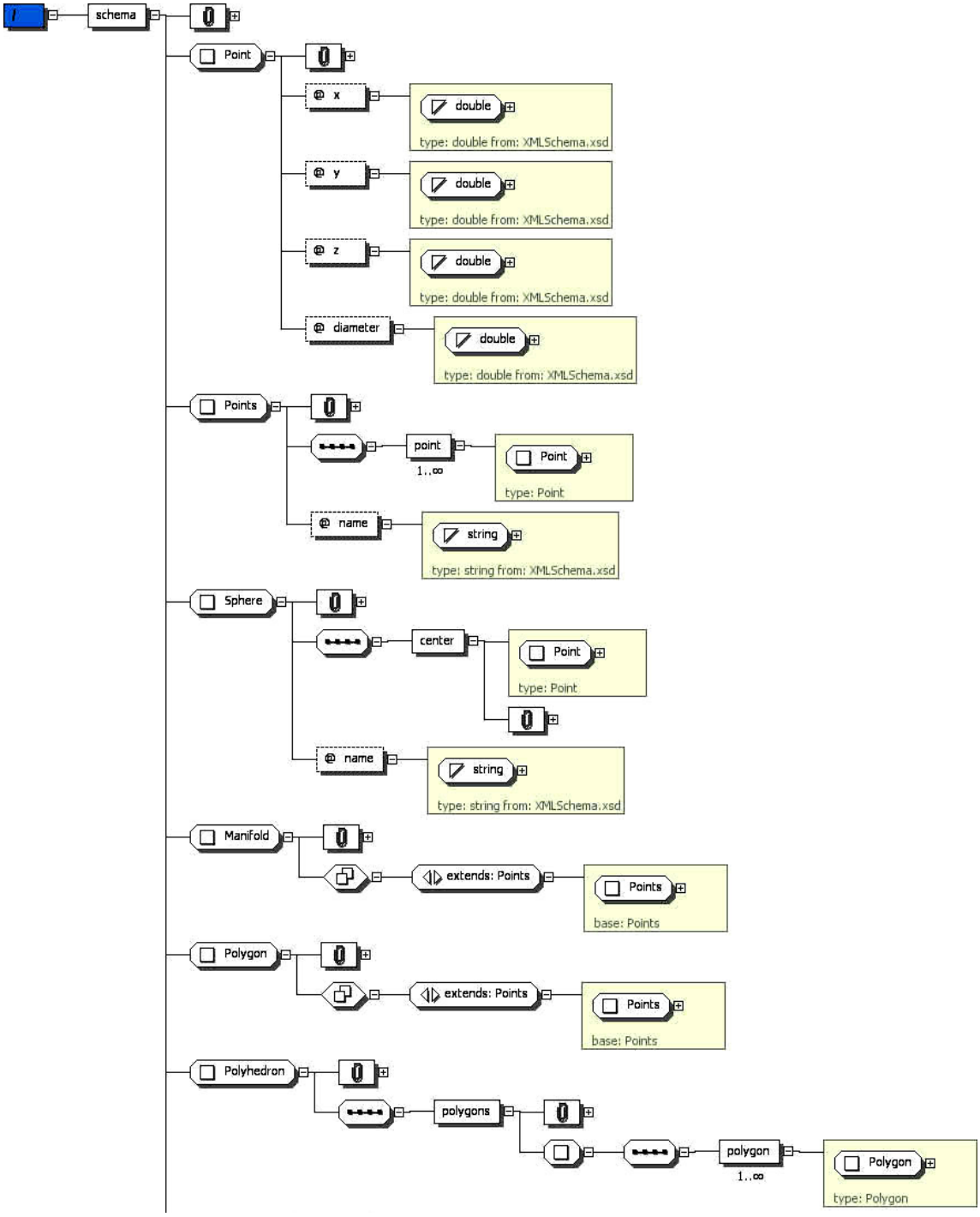


Figure 9: Metadata v1.1 part A

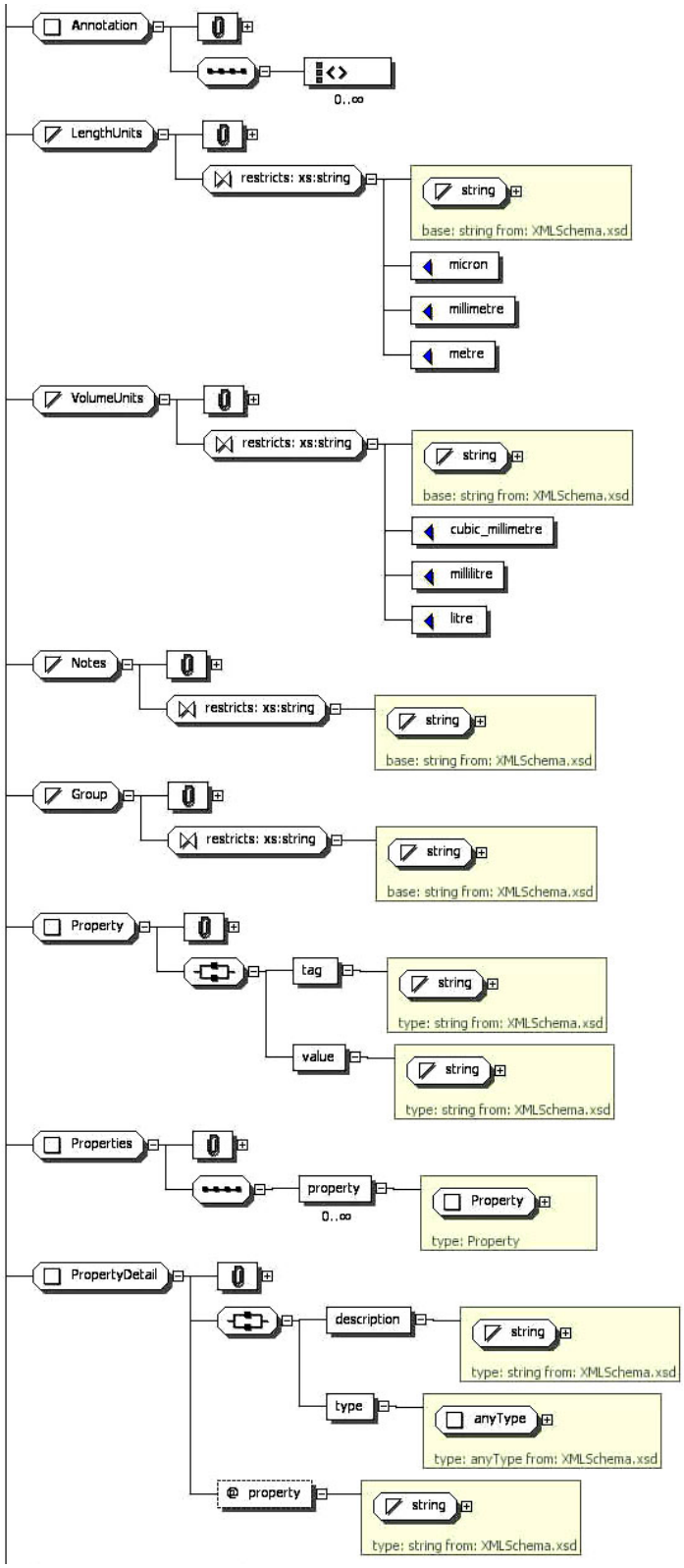


Figure 10: Metadata v1.1 part B

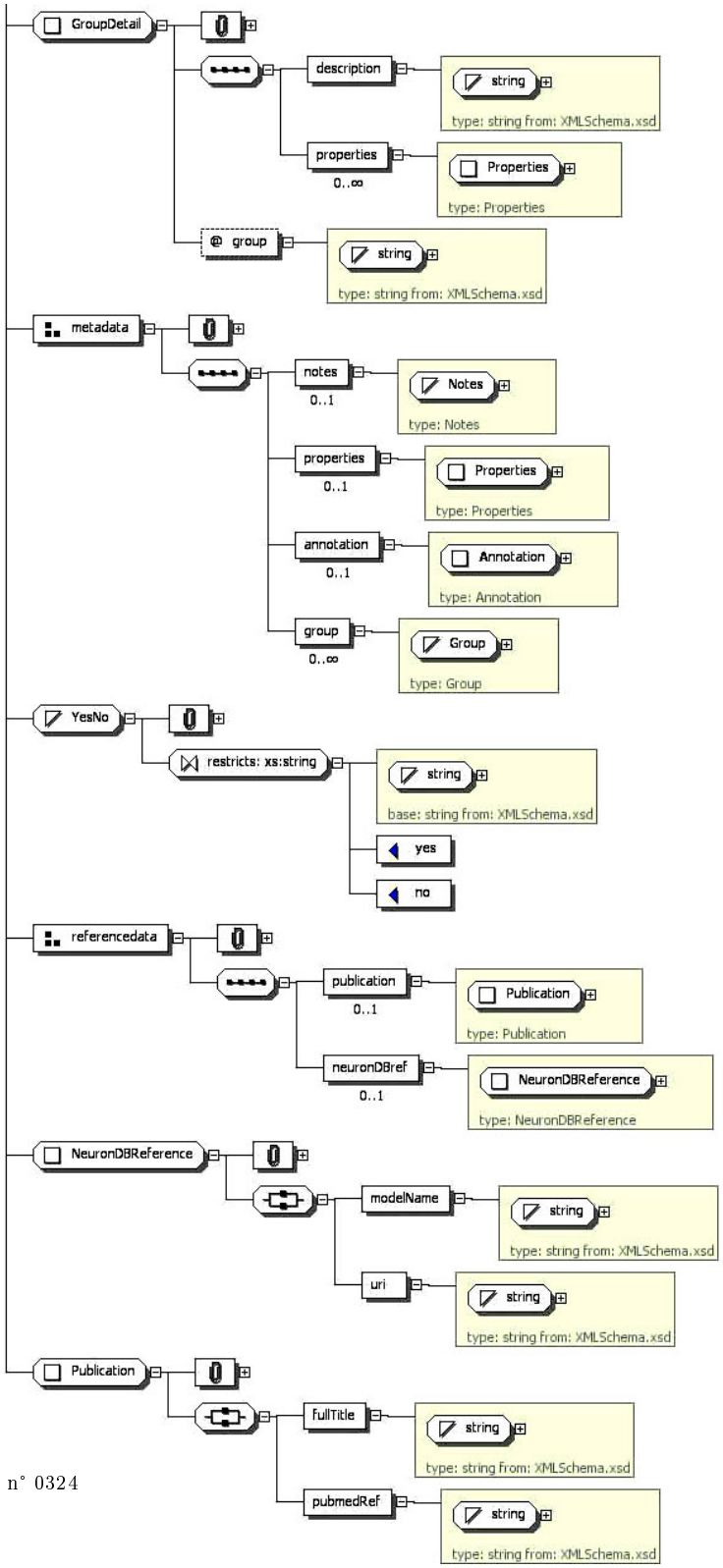


Figure 11: Metadata v1.1 part C

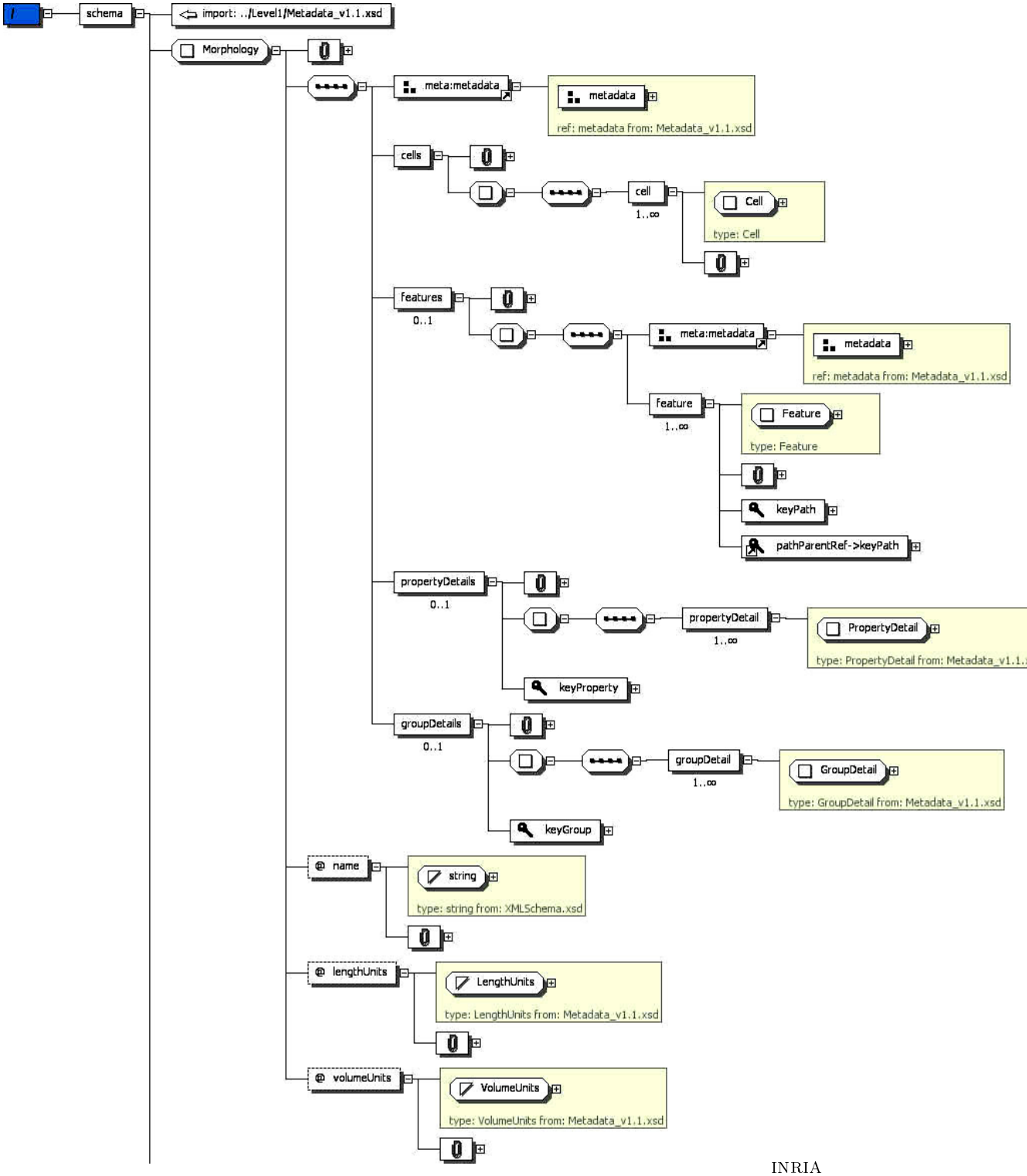


Figure 12: MorphML v1.1 part A

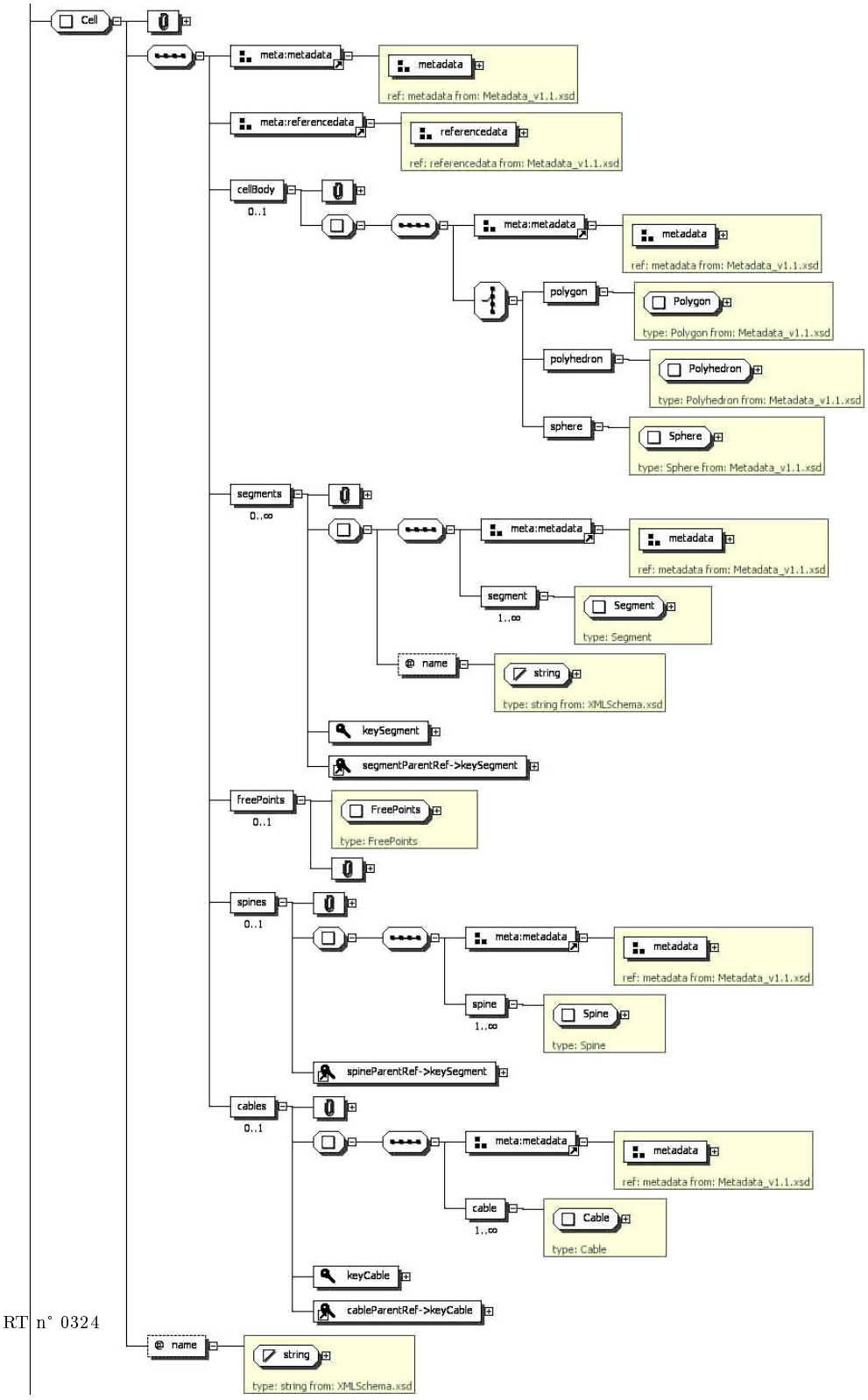


Figure 13: MorphML v1.1 part B

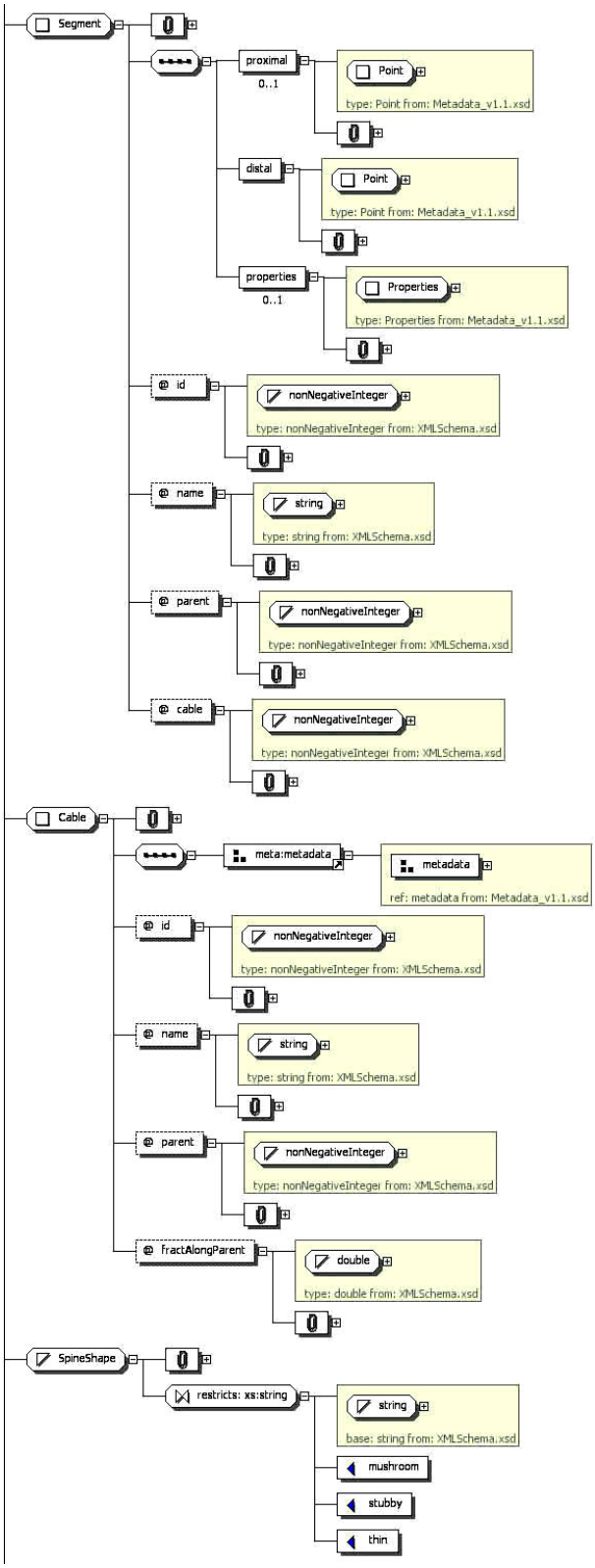


Figure 14: MorphML v1.1 part C

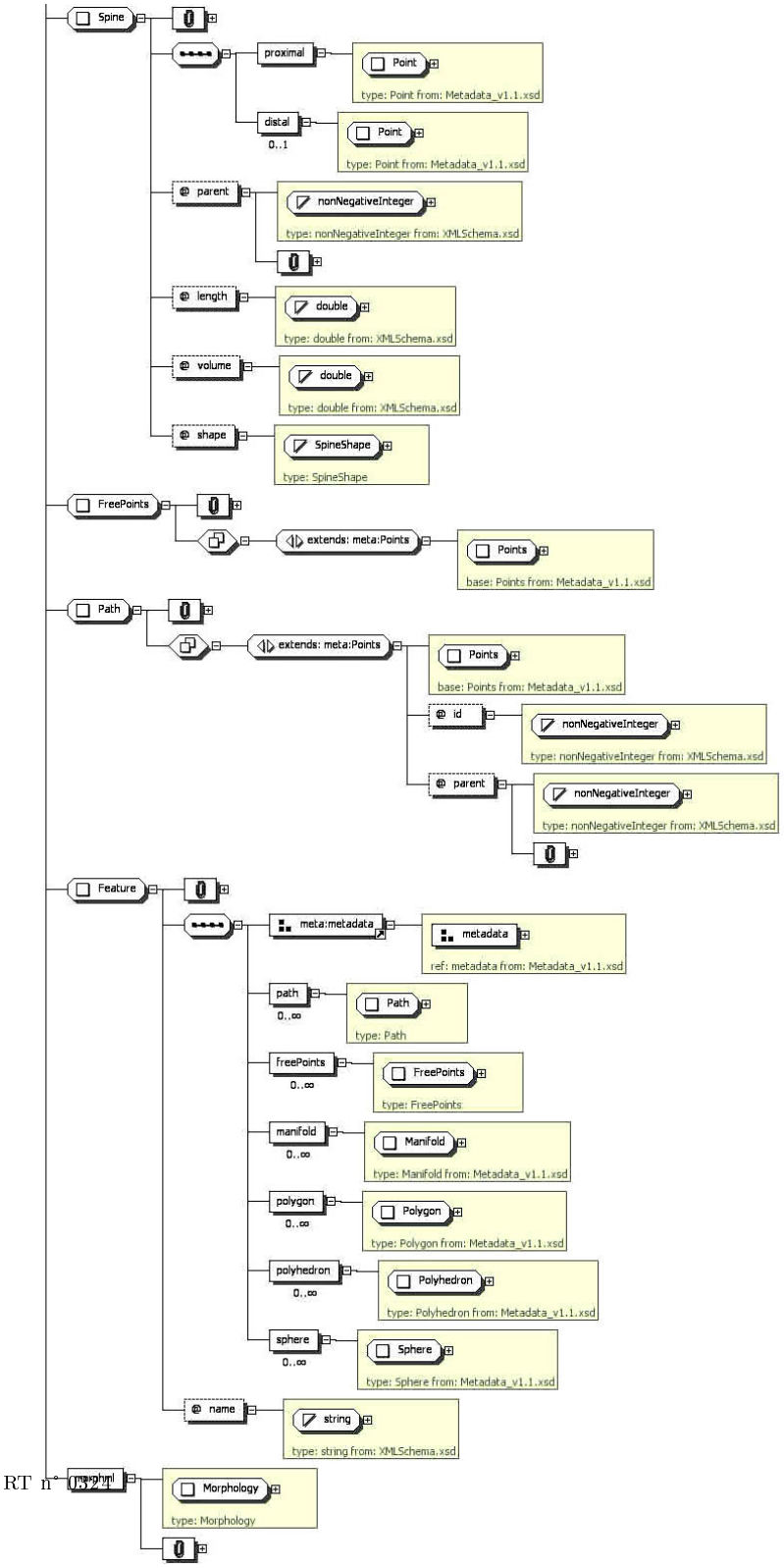


Figure 15: MorphML v1.1 part D

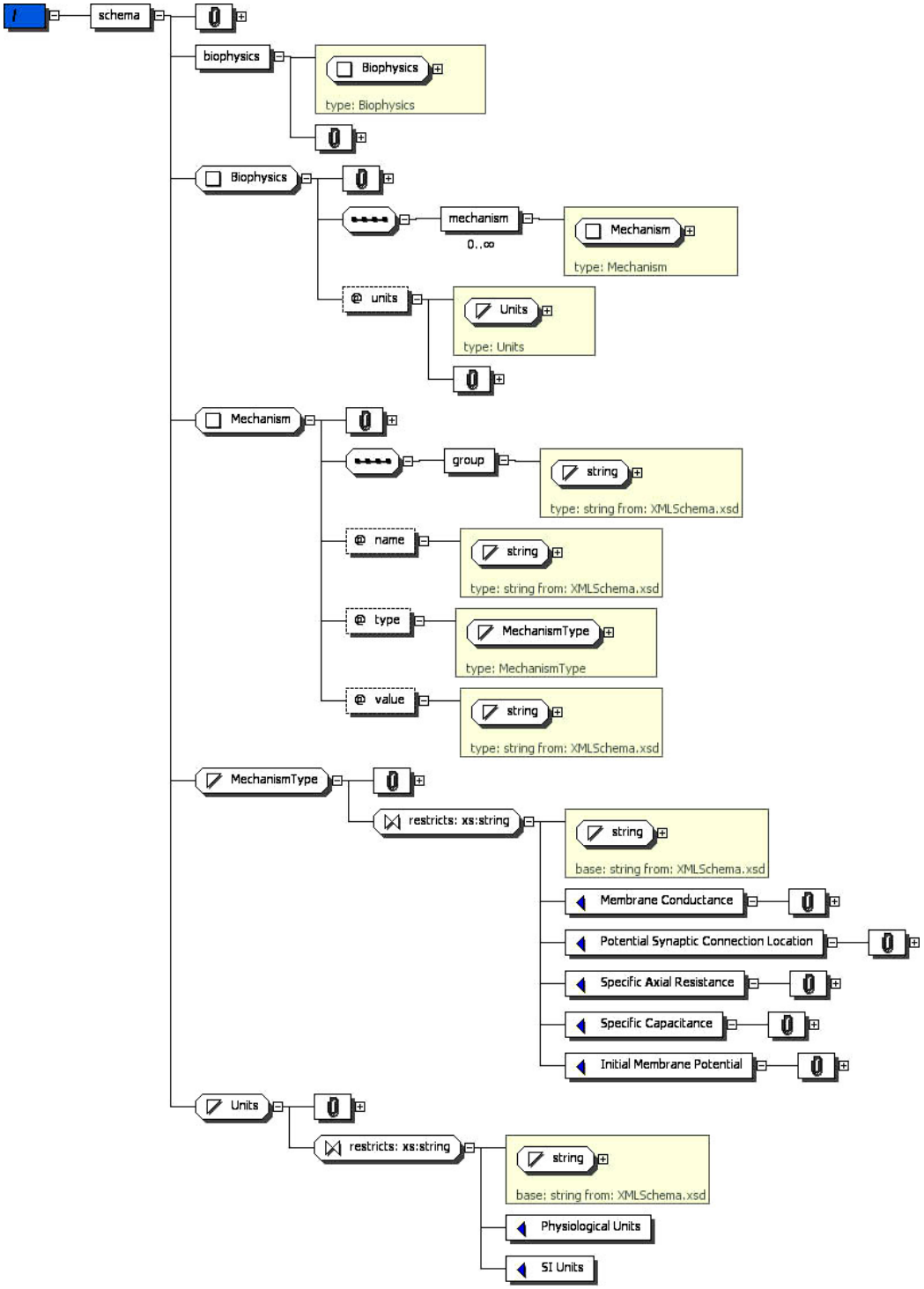


Figure 16: Biophysics v1.1

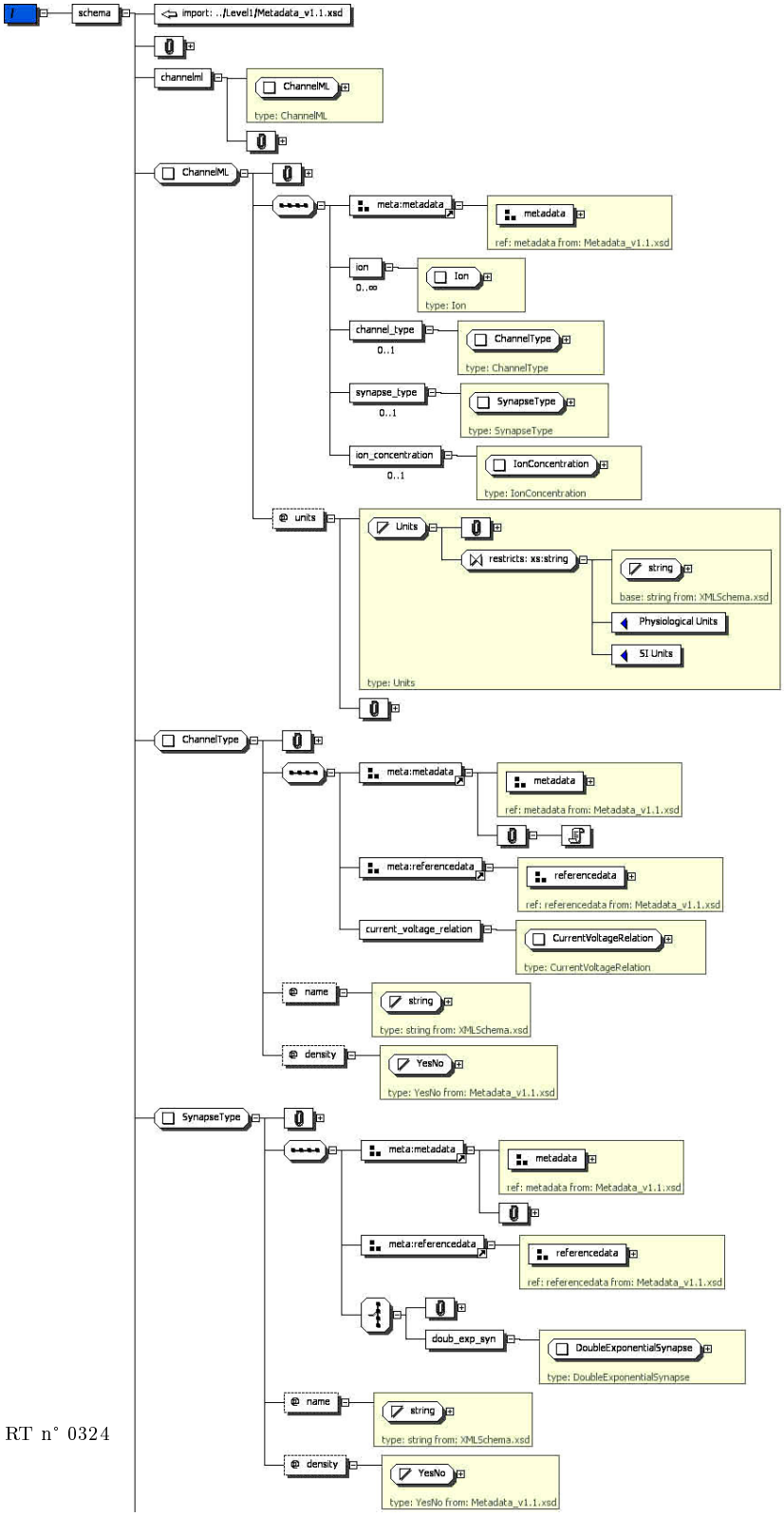


Figure 17: ChannelML v1.1 part A

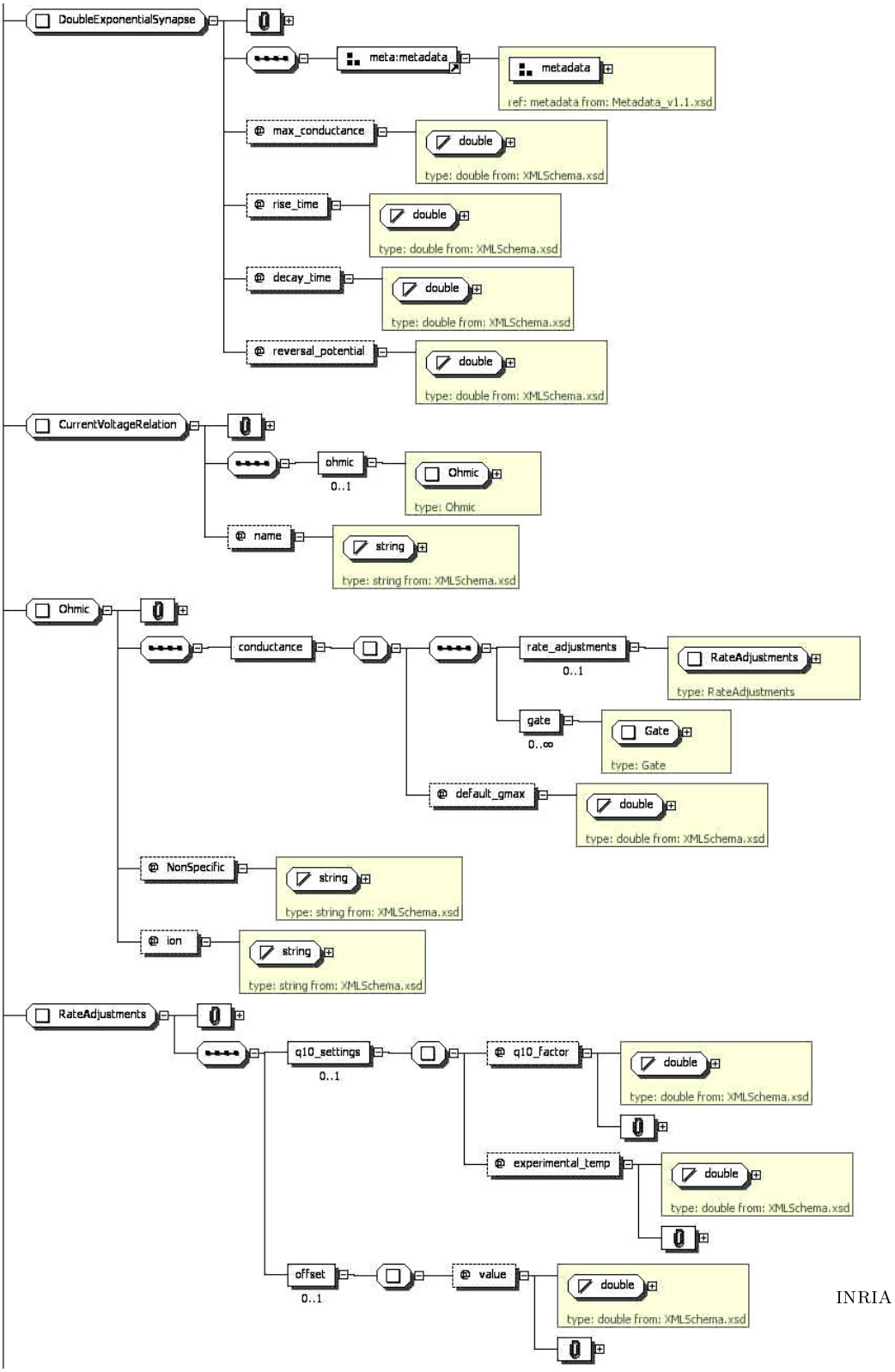


Figure 18: ChannelML v1.1 part B

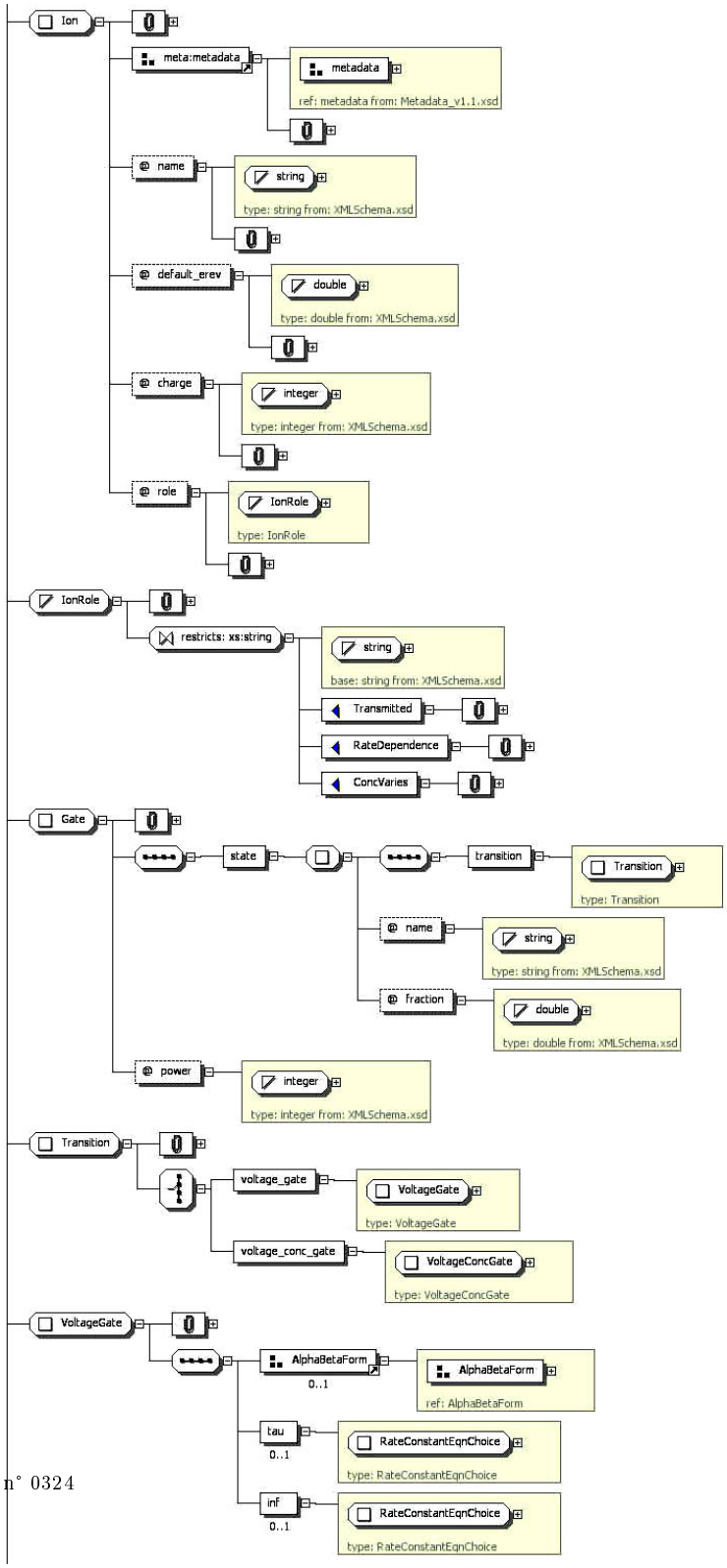


Figure 19: ChannelML v1.1 part C

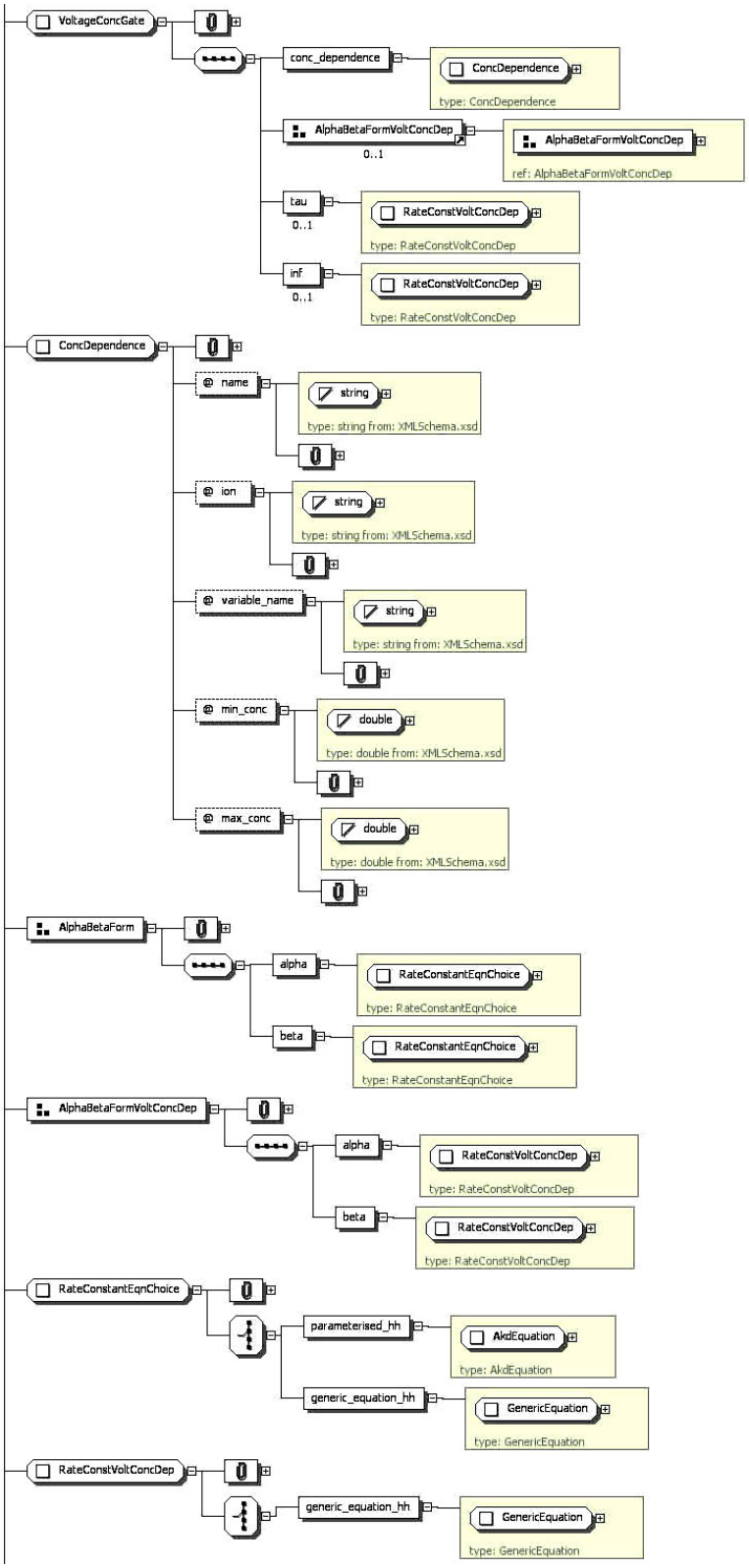


Figure 20: ChannelML v1.1 part D

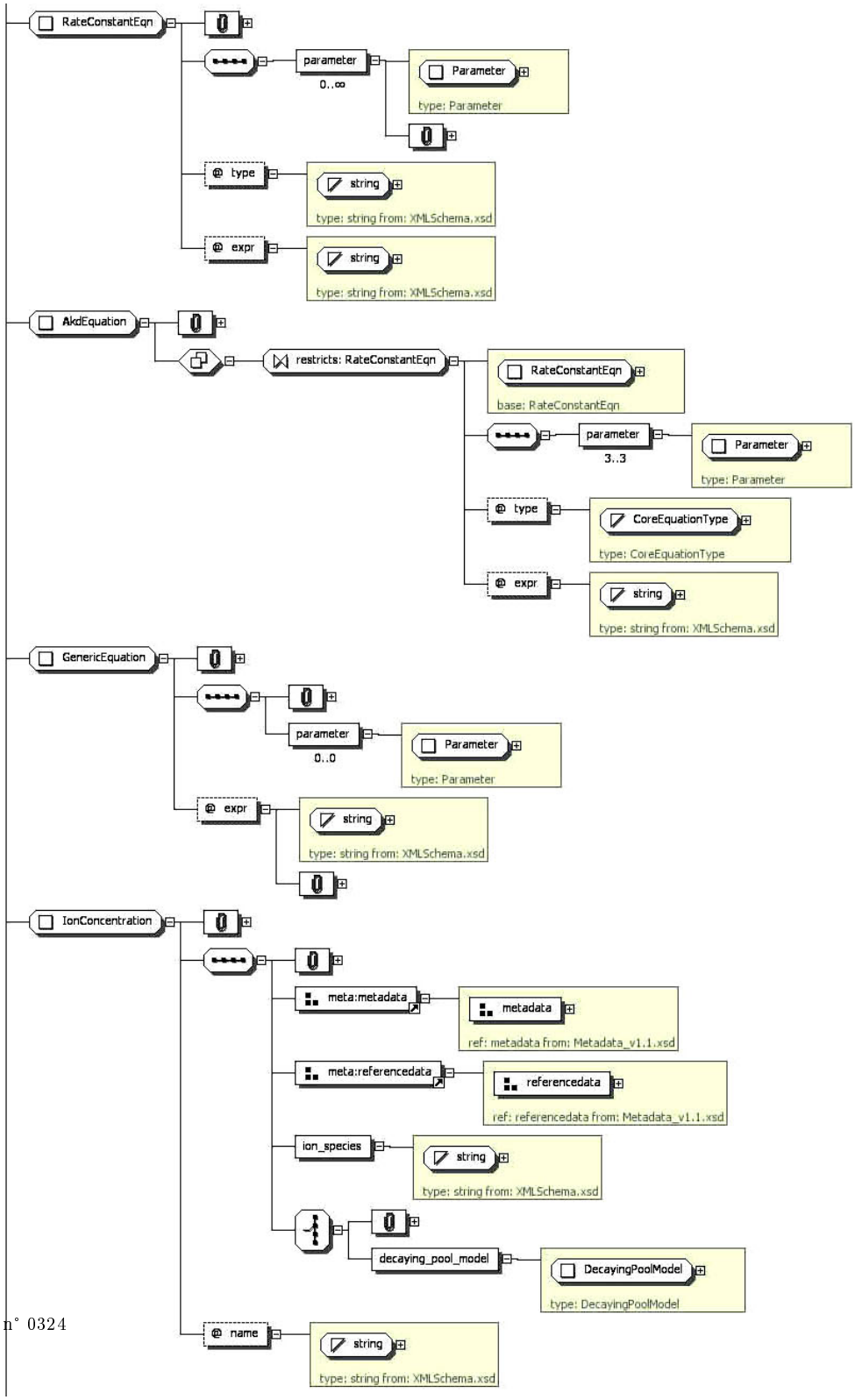


Figure 21: ChannelML v1.1 part E

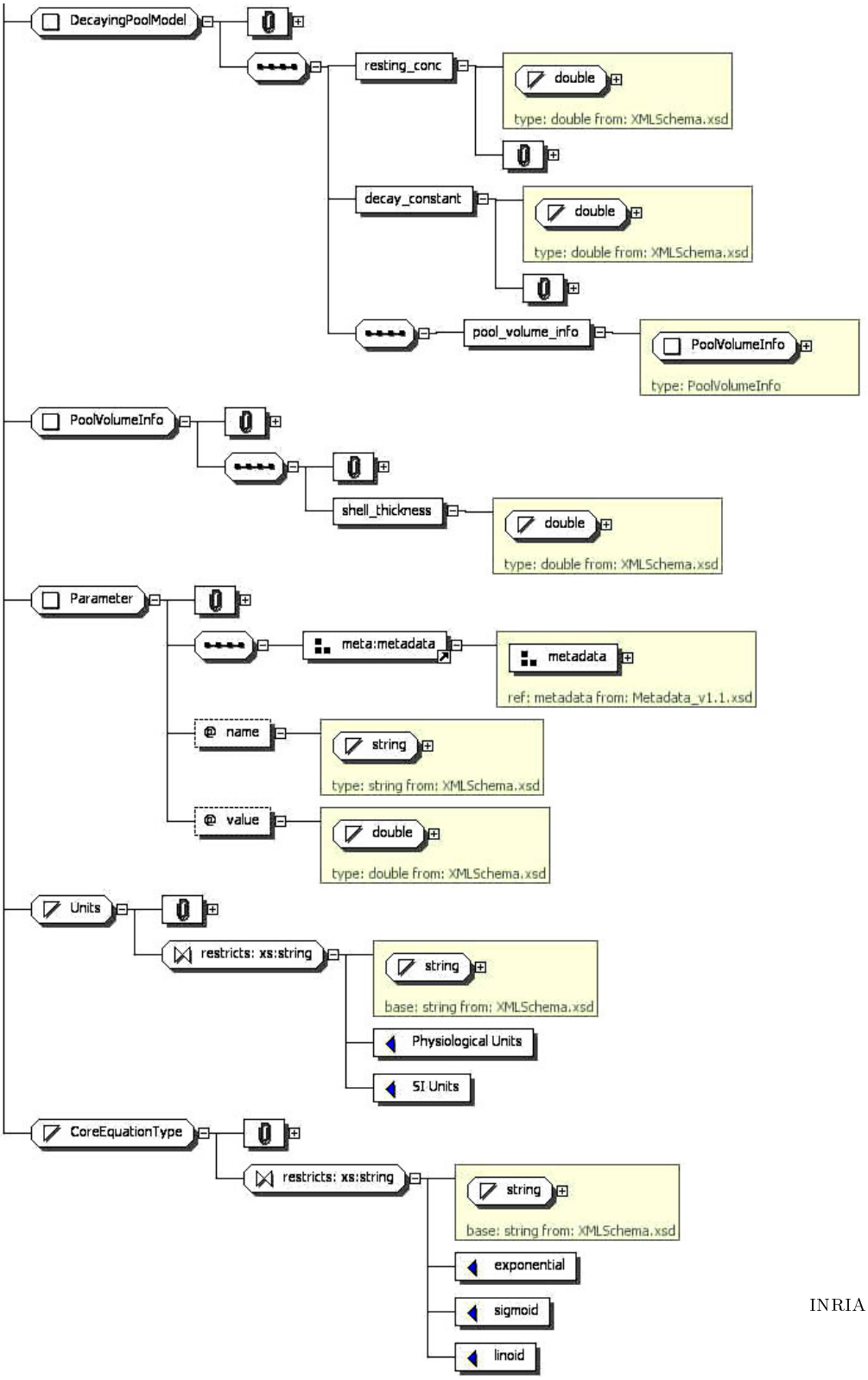


Figure 22: ChannelML v1.1 part F

A RELAX NG alternative
Although XML Schema technology is a worthy and valuable tool, NeuroML requirements for model description could motivate a partial/total change or an enrichment of this schema validating language. In annex a wide portion of the original NeuroML XML Schemata has been translated into RELAX NG, granting quite a complete example.

A simplified list of computational and technical traits which are important to NeuroML will help analysing the RELAX NG alternative (and the Schematron contribution) to W3C XML Schemas :

- to express complex structures (composition of elements, inheritance, ...), since systems in neuroscience are articulated and heterogeneous
- to constraint
  1. datatypes
     (a) on elements
     (b) on element attributes
  2. occurrences of a component
- to maintain unchanged the semantic of each element
- to maintain (as far as possible) clarity in the concretization process (without loosing expression power and correctness)
- to constraint occurrence order of components
- to be able to develop a documentation for biological elements
- to express/reference unique elements
- the easiness of processing data for a translation to/from a different format (for instance GENESIS or NEURON simulation files)
- the code factorization (re-usability of previously defined elements)
  1. identification of specific/particular elements
- the evolutivity of the code already produced
  1. easy to modify ?
  2. easy to enrich ?
- the portability
  1. of the sourcecode
  2. of implemented data

One of the main features NeuroML aims to is the accomplishment of a standard in neuroscience. Since XML is a well known standard, we decided to study the XML syntax of RELAX NG and to forget about the compact form. So, portability of data and code is assured by XML itself.

A first approach to RELAX NG shows us the simplicity in conceiving structured types. The definition of a Sphere which possibly contains an attribute name and characterized by a center of type Point would be as follows :

```
<define name="Sphere">
  <element name="center">
    <ref name="Point"/>
  </element>
  <optional>
    <attribute name="name">
      <data type="string"/>
    </attribute>
  </optional>
</define>
```

Datatyping and structure definition are perfectly distinguished processes in RELAX NG, so we can freely combine them without raising the difficulty of the schema conception. Also, we can plug external datatypes from another schema language : in our example we defined the attribute `name` as a string using `datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"` (inserted for instance on the document root element).
In addition to external datatypes, we can exploit regular expressions and `pattern` facets to constraint various aspects of data. A value restricted between 1.0 and 1.9 would be :

```
<param name="pattern">1.[0-9]</param>
```

RELAX NG basic idea is the named pattern : both elements and attributes can be expressed as a pattern. The flexibility and freedom with which we can combine them and the lack of restrictions associated with these combinations permit to define heterogeneous architectures without loosing clarity. Three styles exist for a RELAX NG schema : Russian doll, DTD-like, or content-oriented. The content-oriented option has the best impact on extensibility and consist in defining the content of an object instead of the object itself. Moreover, inheritance and composition can be implemented without lessening or changing the semantic of an element :

```
<define name="BioCell">
  <ref name="Cell"> <!-- defined in MorphML -->
  <ref name="Biophysics"> <!-- defined in Biophysics -->
</define>
```

If we consider a BioCell as a Cell with biophysical data added, we would not be disappointed by this definition. It also improves code evolutivity. In effect, components can be stored inside external grammars, then easily referenced, enriched or redefined :

```
        ...
<include href="./Level1/Metadata_v1.1.rng"/>
<include href="./Level1/MorphML_v1.1.rng"/>
<include href="./Level2/Biophysics_v1.1.rng"/>
<include href="./Level2/ChannelML_v1.1.rng"/>
        ...
<element name="CellA_enriched">
  <ref name="CellA"/> <!-- from MorphML -->
  <attribute name="addedAttribute">
          ...
  </attribute>
</element>
        ...
<element name="CellB"> <!-- already defined in MorphML, redefined here -->
        ...
</element>
```

On top of that, recursive constructions are allowed and even non-determinism! Thus, a set of cells could be considered as a single `CellA` or a `CellA` with one ore more `CellB` :

```
<define name="Cells">
  <choice>
    <ref name="CellA"/>
      <group>
        <ref name="CellB"/>
        <ref name="Cells"/>
      </group>
  </choice>
</define>
```

Simple patterns permits to specify occurrences of an element : if we want to represent a NeuroML element as defined in the XML Schemata proposed, the basic idea would be

```
<define name="NeuroML">
  <a:documentation>Description of neuronal models,
    including biophysics and channel mechanisms.
  </a:documentation>
    <ref name="metadata"/>
    <ref name="referencedata"/>
    <element name="cells">
```

```
      <oneOrMore>
        <element name="cell">
          <ref name="BioCell"/>
        </element>
      </oneOrMore>
    </element>
    <optional>
      <element name="channels">
          ...
      </element>
    </optional>
    <optional>
      <element name="propertyDetails">
          ...
      </element>
    </optional>
  ...
</define>
```

Individual location of each element or attribute is simply constrained by the way they are ordered in the structure description and even text nodes (`<text/>`), being handled as elements, can be placed exactly where desired.

Unfortunately RELAX NG doesn't offer means to define keys or key references, but it is always possible to exploit ID/IDREF from W3C XML Schema. In the next section we'll see how a set of Schematron rules may help defining some constraints of this nature or may help identifying specific elements.

Various implementations of RELAX NG validating parsers exists : many of them are under a BSD license (like Jing) or another open source distribution (like XVIF or Libxml2). Moreover, RELAX NG, is fairly known and quite spread, even if it doesn't attain the same level as W3C XML Schemas.

A Schematron contribution

A universal and perfect tool doesn't exist and probably never will. Sometimes optimal solutions are achieved by simply mixing imperfect or incomplete ones. Schematron strength is the flexibility of combining the six basic elements (others elements only contribute improving user interfaces) it possesses; however a sole Schematron schema would not be convenient, since it doesn't hold any default rule for data or structure description.

The first problem Schematron can help with is delineating co-occurrence constraints. For instance, we could have an element parameter with a maximal value permitted inside an XML document :

```
<parameter name="A" maxvalue="500" value="432"/>
```

having a value of that parameter greater than its maxvalue would be an error. An embedded Schematron rule inside an XML Schema or a RELAX NG grammar would permit to express such a constraint very easily. With RELAX NG we could have :

```
        ...
<element name="parameter">
  <sch:pattern name="Test constraints on a parameter"
   xmlns:sch="http://www.ascc.net/xml/schematron">
    <sch:rule context="parameter"
     role="value should not exceed maxvalue">
      <sch:assert test="@value &lt; @maxvalue">
        This is the error message we want to print
        if the test fail. In this case we want a value
        lower than the maximal value.
      </sch:assert>
    </sch:rule>
  </sch:pattern>
        ...
</element>
```

Tests always are XPath expressions, so the only payback to simplicity is that an adequate knowledge of XPath is needed to be able to fully exploit Schematron power.

Uniqueness of an identifier and correctness of a reference towards a certain element are issues Schematron can smoothly deal with. Assuming we have a particular biological component of type bioComp and a set of equations placed in another context but related to bioComp, a check could be :

```
<!-- a RELAX NG grammar -->
<define name="equation">
  <sch:pattern name="Test references of equations"
   xmlns:sch="http://www.ascc.net/xml/schematron">
  <sch:rule context="equation[@biocompref]"
   role="reference must exist">
    <sch:assert test="id(@biocompref)">
      An element <name/> should have an attribute
      referencing an existing element.
    </sch:assert>
  </sch:rule>
```

```
  <sch:rule context="equation[@biocompref]"
   role="referenced element must be of a valid type">
    <sch:assert test="(name(id(@biocompref))='bioComp')">
     Referenced element for <name/> should be
     a bioComp.
    </sch:assert>
   </sch:rule>
  </sch:pattern>
    ...
  <attribute name="biocompref"> ... </attribute>
    ...
</define>
```

Other Schmatron rules could help checking text content style, for example the respect of particular format with the intent of producing an output different than XML (i.e. GENESISstab or NEURONChanBuild or NEURONmod, a human readable form, etc.).

2.2 Manipulation

The other aim of NeuroML is to produce files for simulators. There are several methods to produce files. XSL is a part of XML which transforms an XML into a personal file.

XSL

eXtensible Stylesheet Language is a set of three languages approved by the W3C consortium. There are two formatters (XSLT and XSL-FO) and an addressing language (XPath). XSL work with XML files and format them. Moreover, XSL processors must be found in many applications (e.g. Web-browsers) due to the W3C specifications.

XSLT

eXtensible Stylesheet Language Transformations is a set of templates. Each template makes some transformations. It uses XPath to query the XML file.
An example of a XSLT transformation:

```
the xml file <?xml version="1.0" encoding="ISO-8859-1"?>
<print>
<message>Hello , World!</message>
</print>

the xslt file <?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">

<html>

<head>
<title>A test</title>
<style>
body{background-color: rgb(115,120,122);}
p{color: rgb(185,186,187);
  font-family:sans-serif;
  text-align: center;
  font-size: 24px;}
a{color: rgb(185,186,187);}
</style>
</head>

<body>
        <p>
                <xsl:value-of select="print/message"/>
        </p>
</body>

</html>

</xsl:template>

</xsl:stylesheet>
```

We can see a canvas (in italic here). This canvas determines an HTML file. Then <xsl:value-of select="print/message"/> is a rule to search all the nodes print/message and print the value of it.

XSL/FO
Like XSLT, eXtensible Stylesheet Language File Out produces transformations. It is used to produce printable files. Formatted files may be in postscript (.ps), portable document file (.pdf), or in scalable vector graphics (SVG).

XPath
Xpath is a syntax for addressing portions of an XML document. Take care of XPath is not an XML syntax. It is a small query language used in XSLT and XSL/FO.

XSLT in NeuroML
NeuroML uses XSLT to produce simulator's files (e.g. Genesis, Neuron). Each file must be validated before formatted. Indeed, transformations are allowed in the neuroml validator only if the validation passed.

- NeuroML v1.1
  - human readable
    It produces an HTML file. It uses tables tags.
- ChannelML v1.1
  - GENESIStab
  - NEURONChanBuild
  - NEURONmod
  - human readable
    It produces an HTML file. It uses tables tags.
- MorphML v1.1
  - Prime operation
    - test the well fromeness of the document
    - the file must at least one cell. It is noted into the new file
    - ignore tag `CellBody`
    - ignore tag `freePoints`
    - ignore tag `Spines`
  - GENESIS
    - it must be at least one root segment
  - NEURON
    - it must be at least one root segment
    - if there is no cables/sections, it deals segments as an individual section
    - if not all segments are specified, only specified segments are dealed
  - human readable
    It produces an HTML file. It uses tables tags.

Limitations of their implementation
NeuroML can treat only sub languages files (MorphML etc.). Effectively, there are no translators for files written in NeuroML (gather of informations written in sublanguages). NeuroML 1.1 can do less than it pretends. Moreover, there are translation problems. For example, we have a simple MorphML file containing two cells. Each cell has got segments.

Each segment must have a parent segment except the "segment root". There must be only and only one segment root per cell. But, if I have two cells, I will have two segment root (one per cell). Unfortunately, the translator does not understand this subtlety and the file is not formatted. However, in the notes given, the translator would like translate only one cell and forget the other.

This example shows some important problems. The aim of NeuroML is to extend and gather sublanguages for a meta use. Deplorably, transformations cannot work with those files. It is very unpleasant for the user. He must have two sort of NeuroML files: files he can translate and files he can gather informations.

Recommendations
That's why we recommand some different approach. Instead of using only one processor, we use two (or more) processors.

- first processor: its aim is to produce files treated by another processor. It takes care of the potential validations. It informs the user about what will be effectively treated. Let we see an example. We have a NeuroML file containing MorphML cells extended with Biophysics. It contains also channels. We would like to produce channels in GENESIS. The first processor extracts channels from the source file to a file treated by the second processor. It informs le client about cell will be ignored.
- second processor: it is the gather of current xsl private of controls. It treats directely files from the first processor. It produces comprehensible files for simulators (GENESIS, NEURON)
- alternative processors: they are processors dedicated to specific spots. One can for example use XSL/FO to provide documentation in format pdf or ps, an other can also provide diagrams in 3 dimensions (SVG).

CDuce

CDuce permits a powerful manipulation of XML files, but unlike XSLT it offers a functional approach and aims to perform static verifications on documents. CDuce started from XDuce language (UPenn DB group) and it's based on OCaml programming language.

The whole set of XML objects can be handled as first-class values, i.e. they can be freely used as arguments, function results, etc. In addition, CDuce propose a pattern-matching system, a rich type algebra (subtypes are well recognised), function definitions and polymorphism.
All those mechanisms allows complex extractions and precise definitions with a good versatility and a few lines of code.

Unfortunately, CDuce cannot be defined as a "plug-and-play" application : it has a thorny (but not heavy at all!) environment and needs lots of packages before being installed. An alternative to all those packages may be to use the GODI distribution (http://www.ocaml-programming.de/godi/), which includes mandatory and optional packages. Alas, Windows portability cannot be completely assured because the findlib package (version findlib-1.1.2pl1) cannot surely be installed under cygwin.

The final outcome of our analysis claims that CDuce can be very useful in helping for document manipulation and specification, but it would not be suitable for a complete validation or semantic check. Actually, it has the full power to do it but sometimes discloses a strict grammar. Thus, CDuce (unless well mastered) hasn't the vocation to clearly and easily represent heterogeneous and complex data structures.

Some interesting examples, starting from an easy one, will show how CDuce can conveniently contribute to NeuroML.

First, if we suppose the validation process has been previously performed (with XML Schemata or RELAX NG) we won't need any datatype check. So, the elements we want to manipulate can be defined in a generic way, without paying attention to every undertone, but focusing only on those aspects needed by current context. For instance, there would be no need to delimit a diameter as a float value. A `Point` and a set of `Points`, as described in Metadata, could be as follows :

```
type Point = <point x=String y=String z=String diameter=?String>[ ]
type Points = <points>[ Point+ ]
```

hence, the XML code

```
<points>
    <point x="1.5" y="0.0" z="0.0"/>
    <point x="0.0" y="5.0" z="0.0" diameter="15.9"/>
    <point x="1.5" y="2.6" z="3.7"/>
</points>
```

would be a CDuce object of type Points represented by

```
<points>[
    <point x="1.5" y="0.0" z="0.0">[ ]
    <point x="0.0" y="5.0" z="0.0" diameter="15.9">[ ]
    <point x="1.5" y="2.6" z="3.7">[ ]
    ]
```

and we could freely assign that value to a variable.

As previously affirmed, we may represent the whole structure of NeuroML within CDuce, but it would be neither easy nor interesting. A Point isn't a tag but a type : it shouldn't be seen as a tag with some attributes but only a set of attributes. Consequently, the definitions :

```
type Point = {x=String y=String z=String diameter=?String}
type Points = <points name=?String>[ <point (Point)>[ ]+ ]
```

would partially match because even Points should denote a structure.

Manipulation of XML documents can be easily achieved with a few pattern definitions and a small number of code lines. For instance, a valid MorphML document (see appendix : ManyCells.xml) could contain multiple instances of cells which aren't linked (a simple collection of various cells from different systems). Firstly, we can define a set of basic patterns (describing only the main structure) for a Cell, a Cell with Free Points and a MorphML document :

```
type Cell = <cell name=?String>[ _*
                                 <cellBody ..>[ _* ]?
                                 <segments ..>[ _* ]*
                                 <freePoints ..>[ _* ]?
                                 <spines ..>[ _* ]?
                                 <cables ..>[ _* ]? ]

type CellFreePts = <cell name=?String>[ _*
                                        <cellBody ..>[ _* ]?
                                        <segments ..>[ _* ]*
                                        (* pay attention here *)
                                        <freePoints ..>[ _* ]
                                        <spines ..>[ _* ]?
                                        <cables ..>[ _* ]? ]

type Morphml = <morphml ..>[ _*
                             <cells ..>[ (Cell)+ ]
                             <features ..>[ _* ]?
                             <propertyDetails ..>[ _* ]?
                             <groupDetails ..>[ _* ]? ]
```

Secondly, we can simply load the content of the XML document "ManyCells.xml" inside a local variable, specifying to the system that the type should be Morphml :

```
let x :? Morphml = load_xml "ManyCells.xml"
```

Then, we define a very simple function to grasp the list of cells and use it to store the list in a second variable :

```
let getCells(Morphml -> [ (Cell)+ ]) <morphml ..>[ _* <cells ..>x _* ] -> x
let y :? [ (Cell)+ ] = getCells x (* the list of all cells is stored in y *)
```

Finally, we can get the first cell with a set of Free Point, if one exists :

```
(* This function takes a list of cells and gives the first cell which... *)
(* ...contains a set of free points *)
(* If the list is empty or it doesn't contain any CellFreePts, a simple... *)
(* ...error will be raised *)
let fun getCellWithFreePoints ([ (Cell)* ] -> Cell)
  | [ ] -> raise "No Cell with Free Points present"
  | ( [ c ] @ l ) -> match ( [ c ] @ l ) with
      | ( [ (CellFreePts) ] @ [ (Cell)* ] ) -> c
      | _ -> ( getCellWithFreePoints l )

let theCellWeWant :? CellFreePts = getCellWithFreePoints y
```

Input and output are simply but fairly managed : the desired cell will be saved in a XML document by simply writing

```
let head :? String = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
(* the file "output.xml" will be a well formed XML file *)
dump_to_file_utf8 "output.xml" (head @ (print_xml_utf8 res))
(* but pay attention : no line breaks inside the file *)
```

CDuce can really be used as a complementary tool for XSLT. Functions and pattern definitions permits filtering and transforming processes. If we need the list of all parameters (name plus value) to be printed in a human readable format (for instance an html file), we can add some helpful patterns

```
type NamedParam = <parameter name=String value=String>[ ]
type HH = <parameterised_hh ..>[ (NamedParam)+ ]
type HHList = <hh_list ..>[ (HH)+ ]
type L = <li>[ PCDATA ]
type U = <ul>[ (L)+ ]
```

then we can find the list of all parameterised_hh contained in the document, similarly to the way we grabbed the list of cells in the previous example, and store it in a variable, say HHList. The final step would be to define the simple transforming function

```
let fun printParamList(HH -> U ; NamedParam -> L )
    | <parameter name=n value=v>_ -> <li>(n @ " = " @ v)
    | <parameterised_hh ..>e -> <ul>(map e with y -> printParamList y)
```

and apply it in order to obtain the needed result :

```
let result = (map HHList with x -> printParamList x)
```

which could be something like the following CDuce XML object (from file NaF_Chan.xml) :

```
[ <ul>[
    <li>[ 'A = 1500' ]
    <li>[ 'k = 81' ]
    <li>[ 'd = -0.039' ]
    ]
  <ul>[
    <li>[ 'A = 1500' ]
    <li>[ 'k = -66' ]
    <li>[ 'd = -0.039' ]
    ]
  <ul>[
    <li>[ 'A = 120' ]
    <li>[ 'k = -89' ]
    <li>[ 'd = -0.05' ]
    ]
  <ul>[
    <li>[ 'A = 120' ]
    <li>[ 'k = 89' ]
    <li>[ 'd = -0.05' ]
    ]
]
```

A benchmark between CDuce and a fast XSLT processor (performed by the developers, http://www.cduce.org/bench.html) suggests an interesting result : "[...] despite the overhead for static type verification, a CDuce program can run faster (30% to 60%) than an equivalent XSLT style-sheet [...]".

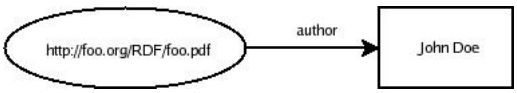
2.3 Specifications

Until now, our works were based on the NeuroML specifications. We have shown what are the problems and what were the solutions. From now on, we suggest a new way of specifications. This new way is based on web-semantic. What is the web-semantic? It is current thought about specifications on the web. Some tools are developed and approved by the W3C to give more interoperability between softwares. Indeed, many applications use their own sub-language. These must have a translator to communicate them selves. The web-semantic show a new interest. Rather using translators, we must communicate with a metadata language. It is not a gather of all sublanguages into a monolithic one, but it is a gather of meta-informations.

RDF
Resource Description Framework is a model of graphs to describe metadata and allow au-

tomatic tasks. These graphs are translated in RDF/XML developed and approved by the W3C consortium. The RDF/XML syntax is now common on web-browsers. There are two projects on the web written with RDF: Dublin-core and RSS. Dublin-core is a project to describe documents and their relationship. For example, a ressource is a book. It has an author, a date of publication etc. This can be described by Dublin-core. The other known project is RSS (RDF Site Summary or Really Simple Syndication). It provides contents or summary on the web. It is often use on web-sites. RSS is a good example of automatic tasks. There are now RSS-reader in browsers or mail-box and they are in some other applications.

The RDF metadata model is based upon the idea of making statements about resources. Theoretically, each datum can be described by a ressource. RDF is a set of triples. These triples define a subject, a predicate and an object. The subject is the resource who it described. The predicate is a relationship between the subject and the object. The object is a value or another ressource. Let see a simple example, as shown in the next figure. The



triple is :

subject http://foo.org/RDF/foo.pdf (a ressource)

predicate author

object John Doe (here a value)

The translation in RDF/XML:

```
<?xml version="1.0"?>

<rdf:rdf  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:f="http://foo.org/schema/">
        <rdf:Description  about="http://foo.org/RDF/foo.pdf">
                <f:author>John Doe</f:author>
        </rdf:Description>
</rdf:rdf>
```

RDF can be exprimed with the first-order logic.
rdfs:subPropertyOf ->
forall s,p1,p2,o T(s, p1, o) and T(p1, rdfs:subPropertyOf, p2) => T(s, p2, o)
forall p1,p2,p3 T(p1, rdfs:subPropertyOf, p2) and T(p2, rdfs:subPropertyOf, p3) => T(p1, rdfs:subPropertyOf, p3) *transitivity*
forall p not T(p, rdfs:subPropertyOf, p) *no cycles*

rdfs:subClassOf ->
forall i, c1, c2 T(i, rdf:type, c1) and T(c1, rdfs:subClassOf, c2) => T(i, rdf:type, c2)
forall p1,p2,p3 T(p1, rdfs:subClassOf, p2) and T(p2, rdfs:subClassOf, p3) => T(p1, rdfs:subClassOf, p3) *transitivity*
forall p not T(p, rdfs:subClassOf, p) *no cycles*

rdfs:range ->
forall p, r1, r2 T(p, rdfs:range, r1) and r1 != r2 => not T(p, rdfs:range, r2)

rdfs:domain ->
forall s, p, o T(s, p, o) and exists d T(p, rdfs:domain, d) => exists d' (T(p, rdfs:domain, d') and T(s, rdf:type, d))
forall s, p, o, r T(s, p, o) and T(p, rdfs:range, r) => T(o, rdf:type, r)

rdfs:literal ->
denote set of literals, declared as a class.

NeuroML specifications according to RDF Let we describe a MorphML simple cell with RDF. It contains only one cell:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:mml="http://morphml.org/RDF/schema#">
   <rdf:Description
     rdf:about="http://morphml.org/RDF/sample/simple_cell">
      <rdf:comment>A simple cell</rdf:comment>
      <mml:Name>Simple cell</mml:Name>
   </rdf:Description>
</rdf:RDF>
```

Here we have an empty cell. mml is a hypothetic namespace. It is not implemented for this example. `http://morphml.org/RDF/sample/simple_cell` is the ressource for our cell. It is also hypothetic.

A cell has got segments and cables :

```
<rdf:Description
  rdf:about="http://morphml.org/RDF/sample/simple_cell">
   <rdf:comment>A simple cell</rdf:comment>
   <mml:Name>Simple cell</mml:Name>
   <mml:segments
```

```
    rdf:ressource="http://morphml.org/RDF/sample/simple_cell#segments"/>
  <mml:cables
    rdf:ressource="http://morphml.org/RDF/sample/simple_cell#cables"/>
</rdf:Description>

<rdf:Description
 rdf:about="http://morphml.org/RDF/sample/simple_cell#segments">
</rdf:Description>

<rdf:Description
 rdf:about="http://morphml.org/RDF/sample/simple_cell#cables">
</rdf:Description>
```

"Segments" is a sequence of "segment" :

```
<rdf:Description
 rdf:about="http://morphml.org/RDF/sample/simple_cell#segments">
  <rdf:Seq>
    <mml:segment
      ressource="http://morphml.org/RDF/sample/simple_cell#segment_1"/>
    <mml:segment
      ressource="http://morphml.org/RDF/sample/simple_cell#segment_2"/>
  </rdf:Seq>
</rdf:Description>

<rdf:Description
 rdf:about="http://morphml.org/RDF/sample/simple_cell#segment_1">
  <mml:Name>Soma</mml:Name>
  <mml:Cable
    ressource="http://morphml.org/RDF/sample/simple_cell#cable_0"/>
  <mml:proximal
    ressource="http://morphml.org/RDF/sample/simple_cell#distal_1"/>
  <mml:distal
    ressource="http://morphml.org/RDF/sample/simple_cell#proximal_1"/>
</rdf:Description>

<rdf:Description
 rdf:about="http://morphml.org/RDF/sample/simple_cell#distal_1">
  <mml:x>0</mml:x>
  <mml:y>0</mml:y>
  <mml:z>0</mml:z>
  <mml:diameter>10</mml:diameter>
</rdf:Description>

<rdf:Description
```

```
 rdf:about="http://morphml.org/RDF/sample/simple_cell#proximal_1">
  <mml:x>10</mml:x>
  <mml:y>0</mml:y>
  <mml:z>0</mml:z>
  <mml:diameter>10</mml:diameter>
</rdf:Description>
```

This is the full description of a segment root.

Now, let we see the description of the second segment :

```
<rdf:Description
 rdf:about="http://morphml.org/RDF/sample/simple_cell#segment_2">
  <mml:Name>Soma</mml:Name>
  <mml:Cable
   ressource="http://morphml.org/RDF/sample/simple_cell#cable_1"/>
  <mml:proximal
   ressource="http://morphml.org/RDF/sample/simple_cell#distal_2"/>
  <mml:distal
   ressource="http://morphml.org/RDF/sample/simple_cell#proximal_2"/>
  <mml:parent
   ressource="http://morphml.org/RDF/sample/simple_cell#segment_2"/>
</rdf:Description>

<rdf:Description
 rdf:about="http://morphml.org/RDF/sample/simple_cell#distal_2">
  <mml:x>10</mml:x>
  <mml:y>0</mml:y>
  <mml:z>0</mml:z>
  <mml:diameter>3</mml:diameter>
</rdf:Description>

<rdf:Description
 rdf:about="http://morphml.org/RDF/sample/simple_cell#proximal_2">
  <mml:x>20</mml:x>
  <mml:y>0</mml:y>
  <mml:z>0</mml:z>
  <mml:diameter>3</mml:diameter>
</rdf:Description>
```

The second segment has a segment-parent which is the segment root.

About RDF The range of RDF is very interesting for NeuroML although it would be necessary to re-examinate all the current specifications. At present, NeuroML conglomerate three main sublanguages. I am not sure it is really clean and exempt of errors. On one hand, NeuroML must be written again with RDF, and also all sublanguages must be specified with

this tool. That induces all the users to reconsider all their files. On the other hand, the structure will be clearly, semantically defined, and would be useful for automatic tasks. Thus, NeuroML would not be any more that one standard of specifications and several satellites would work with these specifications to provide for example data bases (useful to contain billions cells), files of documentation, graphs in 2d or modelings in 3d.

2.4 Overture towards NeuroML v2.0

The comprehensive purpose of each tool is not to be used apart from the others, but to achieve an organic mechanism constituted by a set of interchangeable components.

As previously explained, current NeuroML version gathers and redirects different specializations of biological languages. This "agglomerate" leads to a set of complex mechanisms with multiple possible solutions. RDF is one of them. This solution has a heavy overhead, because it would mean redefining all the sub-languages. Also, we should adapt all manipulation tools. Moreover, we would be pushed to think about a new way to represent NeuroML, with a completely different conception. Benefits would be flexibility and more automated tasks. [[Figure 1.23]] shows a group of possible paths representing almost equivalent solutions.

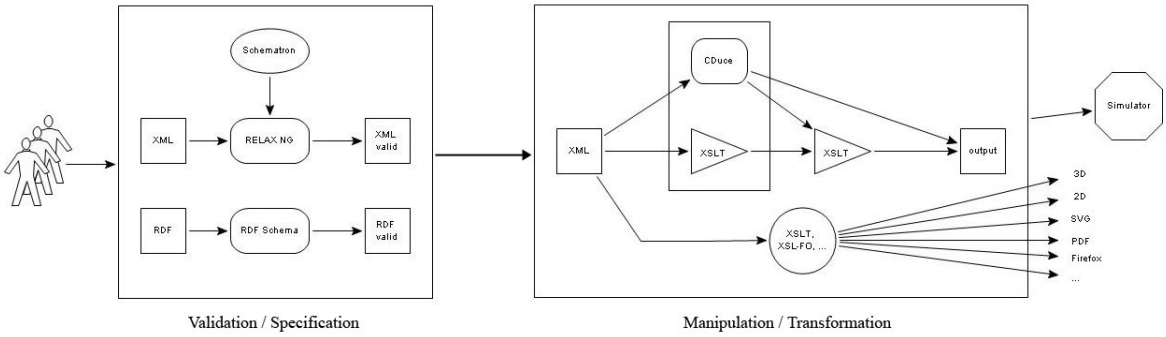


Figure 23: Possible solutions

An interesting solution Resolving paths are meant to be almost equivalent each other because they lead to the same "result" through a finite set of operations. Indeed, they aren't equals. Choosing a solution rather than another could simplify (or complicate or extend) the current step but complicate (or simplify or reduce) the next one.

The following example shows the full mechanism behind one of the possible itineraries. Among all possible solutions, it represents quite a rich one. The main facet of the overall process is to provide an XML portable document between two consecutive steps, as shown in the [[Figure 1.24]].

The first phase would be the validating one : a syntax check, a structural validation and a partial/global semantic examination are performed using REALAX NG grammars enriched by a collection of Schematron rules (helping for complex structural/type constraints

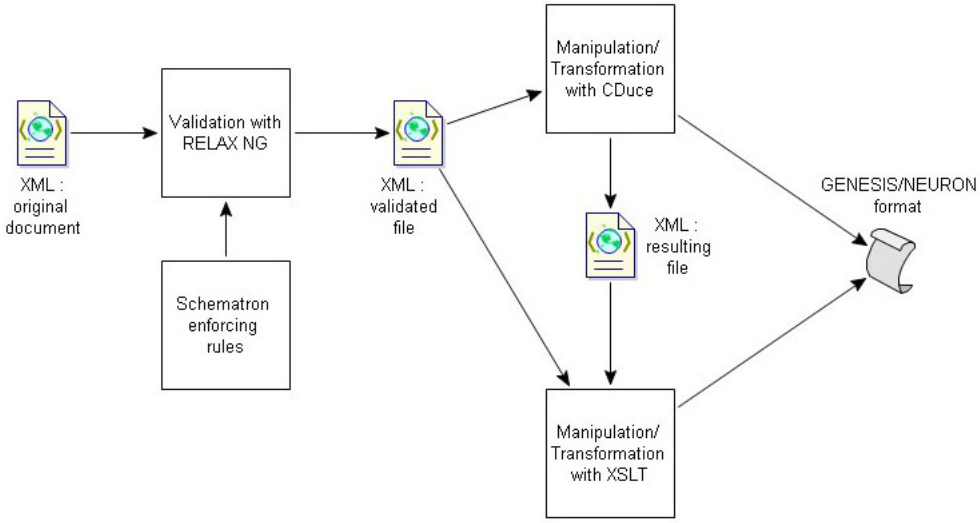


Figure 24: A possible solution using CDuce

and semantic assertions). Once this task has been accomplished, we can be sure of content architecture and datatypes described, so that the following manipulations are lightened and must only focus on specific duty.

The starting document could be as shown by the file "MultipleCells2.xml" given in annex. Even after being validated, the file cannot be directly translated to a GENESIS / NEURON simulation sheet : it is a NeuroML document containing multiple cells with biophysical properties each one. It can be freely shared and stored in a database, but we only want the Mossy Cell to be simulated. This cell should drop biophysical data before simulation. Moreover, it would be useful to add some descriptive contents.

Thus, the second phase is split into two sub-phases : the manipulating one and the translating one. Instead of using a monolithic set of XSL transformations, CDuce will help extracting and modifying the Mossy Cell. XSLT will only have to care about translation, forgetting about unnecessary data.

We can start defining some CDuce pattern : a biological cell structure, a segment, the structures of the cell we need and the resulting cell we want to simulate, the generic shape of a NeuroML document. The following code is a bit more verbose than necessary, with the double intent of showing some CDuce syntactical traits and expressing the main ideas.

```
type BioCell = <cell name=?String>[ _*
                                    <cellBody ..>[ _* ]?
                                    <segments ..>[ _* ]*
                                    <freePoints ..>[ _* ]?
                                    <spines ..>[ _* ]?
                                    <cables ..>[ _* ]?
                                    <biophysics ..>[ _* ]
                                    ]

type Segment = <segment id=String cable=?String parent=?String>[ _* ]

type NeededCell =<cell name="MossyCell">[ _*
                                          <segments>[ (Segment)* ]
                                          <freePoints ..>[ _* ]?
                                          <spines ..>[ _* ]?
                                          <cables ..>[ _* ]
                                          <biophysics ..>[ _* ]?
                                          ]

type NamedSegment = <segment name=String id=String
                             cable=?String parent=?String>[ _* ]

type WantedCell =<cell name="MossyCell">[ _*
                                          <segments>[ (NamedSegment)* ]
                                          <freePoints ..>[ _* ]?
                                          <spines ..>[ _* ]?
                                          <cables ..>[ _* ]
                                          <biophysics ..>[ _* ]?
                                          ]

type Neuroml = <neuroml ..>[ _*
                             <cells ..>[ (BioCell)+ ]
                             <channels ..>[ _* ]?
                             <propertyDetails ..>[ _* ]?
                             <groupDetails ..>[ _* ]?
                             ]
```

Then we need to load the source file and extract the MossyCell.

```
let x :? Neuroml = load_xml "MultipleCells2.xml"

let getCells(Neuroml -> [ (BioCell)+ ]) <neuroml ..>[ _* <cells ..>x _* ] -> x
```

```
let y :? [ (BioCell)+ ] = getCells x

let fun getNeededBioCell ([ (BioCell)* ] -> BioCell)
    | [ ] -> raise "Not Found"
    | ( [ c ] @ l ) -> match ( [ c ] @ l ) with
        | ( [ (NeededCell) ] @ [ (BioCell)* ] ) -> c
        | _ -> ( getNeededBioCell l )
```

Before being simulated, the Mossy Cell should abandon biophysical data.

```
let fun dropBiophysics (NeededCell -> NeededCell)
    | (<cell name=n>x) -> match x with
        ([ _* b & (<segments ..>[ _* ]) <freePoints ..>[ _* ]?
           <spines ..>[ _* ]? c & (<cables ..>[ _* ]) <biophysics ..>[ _* ]? ])
        -> <cell name=n>[ b c ]

let mossy :? NeededCell = dropBiophysics ( (getNeededBioCell y) :? NeededCell )
```

A descriptive enrichment can be performed quite easily. For instance, we can add names with explicit meaning to the set of unnamed segments : a translated sheet (such as GENESIS or NEURON script) would be more human readable.

```
let fun addName(Segment -> NamedSegment)
    | <segment id=i>x -> <segment name=("Segment_" @ i) id=i>x
    | <segment id=i cable=c>x -> <segment name=("Segment_" @ i) id=i cable=c>x
    | <segment id=i parent=p>x -> <segment name=("Segment_" @ i) id=i parent=p>x
    | <segment id=i cable=c parent=p>x ->
        <segment name=("Segment_" @ i) id=i cable=c parent=p>x

let fun addNames (NeededCell -> WantedCell)
    | (<cell name=n>x) -> match x with
        ([ _* <segments ..>y <freePoints ..>[ _* ]? <spines ..>[ _* ]?
         c & (<cables ..>[ _* ]) <biophysics ..>[ _* ]? ])
        -> <cell name=n>[ <segments>(map y with v -> addName v) c ]

let res :? WantedCell = addNames mossy
```

The last step is to save the final cell, ready to be simulated, into a valid MorphML file, as shown by the resulting file "res.xml" given in annex.

Thereby, we must guarantee that dumping XML objects into files preserve validity towards NeuroML schemas.

```
let head :? String = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>
                      <morphml name=\"Unique cell\" lengthUnits=\"micron\">
```

```
                        <cells>"

let tail :? String = "  </cells>
                      </morphml>"

dump_to_file_utf8 "res.xml" (head @ (print_xml_utf8 res) @ tail)
```

Finally, we can use XSL transformation as a simple automatic task (second sub-phase of second phase) to obtain the simulation script.

3 Synthesis

In our report, we have approached a panoply of difficulties related to the use of NeuroML and we have proposed solutions to solve these problems.

The first version (i.e. NeuroML 1.0) used Java support as a mapping tool. This language does not necessarily constitute the best tool for handling XML data :

- object-oriented paradigm permits a fine model and quite a rich specification level, but in the event of misuse it can generate many conceptual and management problems,
- the programming overhead may be high enough, needing compilation before execution.

However, starting from version 1.1, NeuroML uses exclusively XML support. It allows a good specification level without needing a software architecture too complex. Indeed, the overhead is not reduced but only shared between two tools : XML Schemas for the specifications/validation and XSLT for the manipulation/transformations. The cost of management and the level of complexity could be shared later on : introducing RELAX NG and Schematron, share of transformations among several XSLT processors, introducing CDuce, etc. In this context, only RDF would be the true alternative requiring a completely re-examined design.

Prototype

4 Choosing an XML editor

NeuroML people are working with documents in XML format. With the increasing popularity of XML, the number of XML editors is growing exponentially and it can be extremely difficult to choose the convenient editor that suits NeuroML users. The aim of this section is to introduce the different features XML editors have and to present the result of an evaluation exercise we have done trying a large number of editors.

This section first outlines the different types of XML editors that are available and their main characteristics. Then we will expose the different features an XML editor should have to help NeuroML users. Finally, the editor which was preferred by our group and the extension we added to it.

5 Types of editors

An *editor* is a tool used for creating and/or modifying all kinds of text files. it allows a user to write text and to manipulate it in a variety of fashions. There are literally tons of editors available, and we consider that no single one is the best as they all pretend to be!! Peoples choice of what editor to use is often regarded as a religious issue. We are totally

convinced that everyone has a favorite and usually one's choice is influenced by which editor one learned first.

There are 4 main types of XML editors, the simplest are the *Text editors*. There is a number of text editors which have added functionality for XML encoding. These functions often include Syntax highlighting, validation, and sometimes auto-completion of tags and attributes. There is also dedicated XML editors that are software built uniquely for the purpose of encoding XML. Those editors does not offer functionality for other tasks. There are two types of dedicated XML editors. The most common and most popular type of editor is called screen-oriented. There are also sometimes referred to as *visual* XML or *code view* editors. The user can work a view that presents the XML tags as small icons. The second type of dedicated XML editors are *visual* editors combined with word-processors, which will be discussed below. These represent the third type of editor. This type of editor offers the user some sort of **WYSIWYG** or **What you see is what you get**.

Word-processors approach XML from a different angle. They provide the possibility of opening XML documents and working with them in a **WYSIWYG** view that hides the XML code for the user. They can also be used to create a new document and save it as a valid XML file.

6 Features of XML editors

This section introduces a number of features of XML editors. The following list presents twenty five, listed with a brief description, that we've identified as useful by XML encoders. &to be used by NeuroML community.

1. Validation of the XML file within the editor
   The editor should have an option to check whether the document is valid or not. The most interesting thing would be to have a visible button on the editors main frame to validate the XML document.

2. Continuous validation of XML documents
   The editor continuously checks if the right elements are being used. The most interesting thing would be to highlight the erroneous elements in a highly visible color (exemple: red) this option might be called *Text Watcher*.

3. Menu for adding allowed elements and attributes
   The editor provides a menu of elements and attributes which are allowed at that particular point in the text. The most interesting thing would be to introduce an easy-access menu "e-Complete" in the editors menu bar.

4. Auto-completion of elements, attributes and default attribute values
   The editor provides while, one is typing XML tags, a list of possible elements, attributes

as well as attribute initialization. The most interesting thing would be to introduce in the "e-Complete" menu an item that activates this auto-completion option.

5. Template creator
   The editor allows the creation of templates for new documents.

6. Operate on the XML structure
   The editor should provide functions such as copy/cut/paste within the XML document based on its structure. Those options should be introduced in the Menu bar and the right mouse click menu.

7. Text editing features
   The editor provides all of the conventional text editors options i.e. find/replace/select all etc... All of those options should appear in the menu bar, keyboard shortcuts should would be nice to add. The editor should let you try out your ideas or changes and go back to the original with unlimited levels of undo/redo.

8. Syntax highlighting
   The editor uses syntax highlighting when working or viewing the code itself. It would be interesting to be able to activate/deactivate it with a button.

9. View document structure
   The editor has a pane which allows hierarchical display of the entire document.

10. Structure-aware search and replace
    The editor allows the searching and replacement of certain tags. This option should work in concordance with the automatic validaton option. An example search could be : searching the `<a>` tags and replacing them with `<b>` tags :it will automatically replace `</a>` tags with `</b>` tags. The most interesting thing would be to insert this option in the menu bar as well as giving a easy shortcut to activate it.

11. Extensible Spell-checker
    The editor can check the spelling of text in your documents. You can check the spelling for the entire document or a single word, or have the editor check your spelling as you type. It is possible to edit the word lists of the spell checker and to use different languages within one document.

12. Multilingual text input and display (Unicode) The editor can handle different writing systems based on the Unicode character set. Changing the Unicode character set will automatically change the spell-checker to the corresponding language. The most interesting thing would be to have an easy to access small scroll menu containing the different Unicode Sets.

13. Multi-Schema languages support
    The editor supports XML documents based on different schema languages for describing the structure of the XML documents.

14. Built-in XSLT processor
    The editor has a built-in XSLT processor based on NeuroML XSLT stylesheets. It converts XML documents with XSLTstyle sheets into NeuroML format XML documents.

15. XML to PDF conversion (XSL-FO processor)
    The editor has a built-in XSL-FO processor t convert XML documents into PDF documents.

16. Create/Edit/Erase XSLT stylesheets
    The editor supports editing and creating XSLT stylesheets. This is interesting in case NeuroML version changes, having such an option proofs that the editor is extensible.

17. Multiple file management
    The editor can work on several XML documents simultaneously. Operations include validating documents, converting documents with XSLT stylesheets etc ...

18. Multiple synchronized views
    The editor has three type of views :

    - the "code view" is where the user can work on the XML code itself.
    - the "tags on view" is where the user can work on the XML code but can also see tags that are being seen surrounding the content.
    - the "normal view" is where user can work on the XML code without seeing the tags that are used.

19. Version control
    The editor has a built-in functionality to automatically manage the different versions of a document. The most interesting thing would be to start counting versions when the user creates a new document.

20. Working with large documents
    The editor works quickly with large documents. NeuroML files might be extremely large, it would be interesting to be able to open those files quickly and to edit them effectively.

21. Editing documents contained on a FTP/WebDAV
    The editor supports remote editing, using the FTP and WebDAV protocols. The remote files can be added to the project an can be subject to XSLT or XSL-FO transformations.

22. Editing documents contained in a Zip
    The editor supports editing files contained in archives.

23. Easy multi-lingual documentation
    The editor has a easy documentation. It should appear in the menu bar.

24. Easy tutorials/hints
    The editor has an easy tutorial and hints options that can be activated.

25. Multi-platform: Linux, MacOS X, Windows

7 Editors and features

In this section will be presented to you the different features of 4 different editors we've evaluated. We've chosen those particular softwares depending on the survey we'v done on the NeuroML community (http://kommunix.free.fr/neuroml/sondage.html). All evaluations are subjective assessments where availability and usability of each feature in each editor has been taken into account.

SOFTWARE	1. Validation of the XML file within the editor	2. Continuous validation of XML documents	3. Menu for adding allowed elements and attributes	4. Auto-completion of elements, attributes and default attribute values
MS XML Notepad	with Firefox	no	no	no
Serna	yes	yes	yes	yes
XML Mind	yes	?	yes	?
oXygen	yes	yes	yes	yes

SOFTWARE	5. Template creator	6. Operate on the XML structure	7. Text editing features	8. Syntax highlighting
Serna	yes	yes	yes	yes
XML Mind	?	yes	yes	yes
oXygen	yes	yes	yes	yes

SOFTWARE	9. View document structure	10. Structure-aware search and replace	11. Extensible Spell-checker	12. Multilingual text input and display (Unicode)
Exchanger XML	yes	yes	?	?
Serna	yes	yes	yes	yes
XML Mind	yes	yes	yes	yes
oXygen	yes	yes	yes	yes

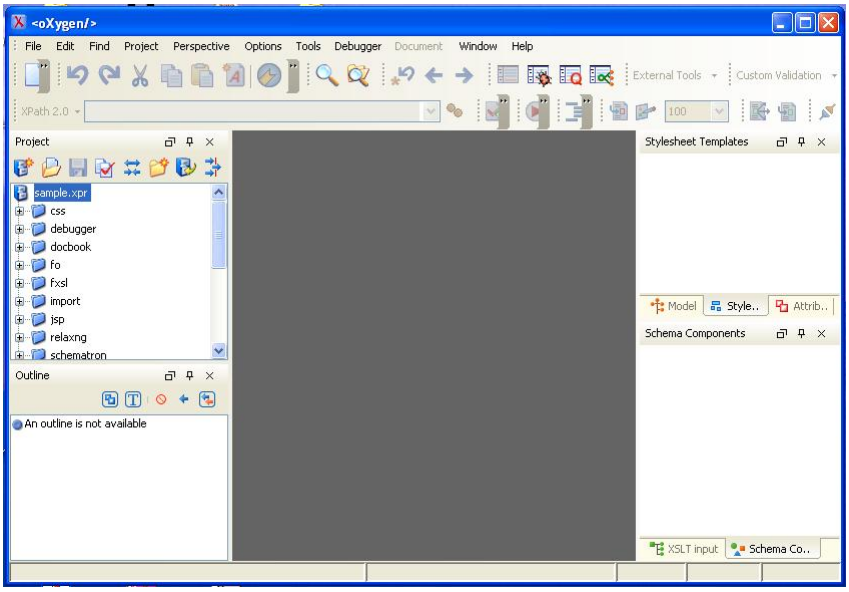
SOFTWARE	13. Multi-Schema languages support	14. Built-in XSLT processor	15. XML to PDF convertion (XSL-FO processor)	16. Create/Edit/ Erase XSLT stylesheets
Exchanger XML	?	yes	yes	?
Serna	yes/no RNG	yes	yes	yes
XML Mind	yes/no RNG	yes	yes	?
oXygen	yes	yes	yes	yes

SOFTWARE	17. Multiple file management	18. Multiple synchronized views	19. Version control	20. Working with large documents
Exchanger XML	?	yes	no	?
Serna	yes	yes	no	yes
XML Mind	yes	yes	no	?
oXygen	yes	yes	no	yes

SOFTWARE	21. Editing documents contained on a FTP/WebDAV	22. Editing documents contained in a Zip	23. Easy multi-lingual documentation	24. Easy tutorials / hints
Exchanger XML	yes	no	?	yes
Serna	yes	no	no	yes
XML Mind	yes	yes	?	yes
oXygen	yes	no	yes	yes

SOFTWARE	25. Multi-platform: Linux, MacOS X, Windows
MS XML Notepad	Mac/Linux/Win
Peter's XML Editor	Win
XML Writer 2.5	Win
XML Spy	Win
Authentic	Win
Epic Editor	Solaris/Win
Morphon	Win/dead website
Alchemist XML IDE	Win
Open Office	Linux/Win
XML Viewer 1.3	Win
XML Fox 1.61	Win
Stylus Studio XML	Win
XmlShell	Win
Cooktop	Win
Exchanger XML	Linux/Win
Serna	Linux/Mac/Win
VIM	Plugin pour VI
NoteTab	Win
XML Mind	Linux/mac/Win
oXygen	Linux/Mac/Win
Altova XML Suite	Win
Stylevision	Win
Emacs	Linux/Mac/Win

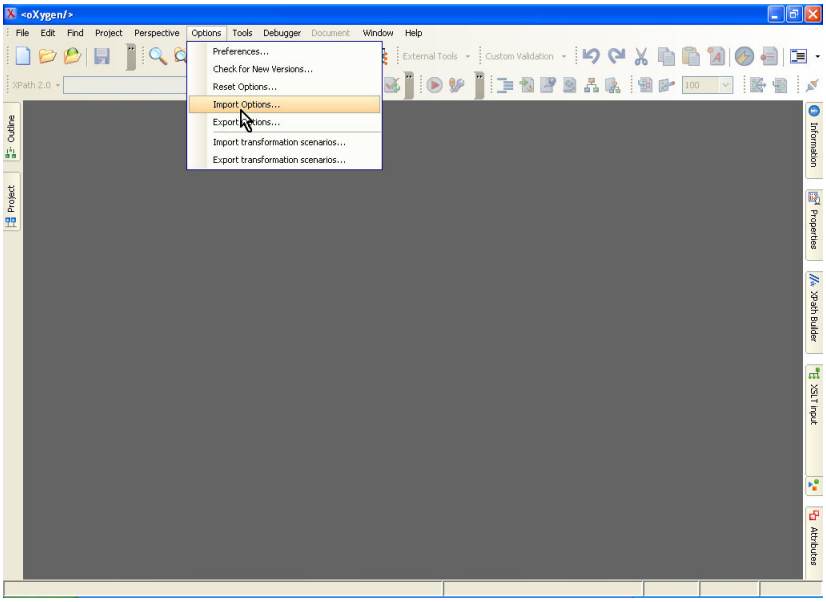
Considering all of those criteria we've chosen <oXygen/> for many reasons. The <oXygen/> 7.0 XML editor is an excellent addition to any professional Web programmer's program suite. A single copy (can be installed on one machine, but documents produced can be used for any purpose) can be purchased for $96 USD which converts to approximately 76€. It has a very well laid out, aesthetically pleasing GUI and, where certain other programs in this category have an old look, this program definitely has that up-to-date quality. The main program window is divided into three sections: the Model, Outline and Code entry windows as illustrated :



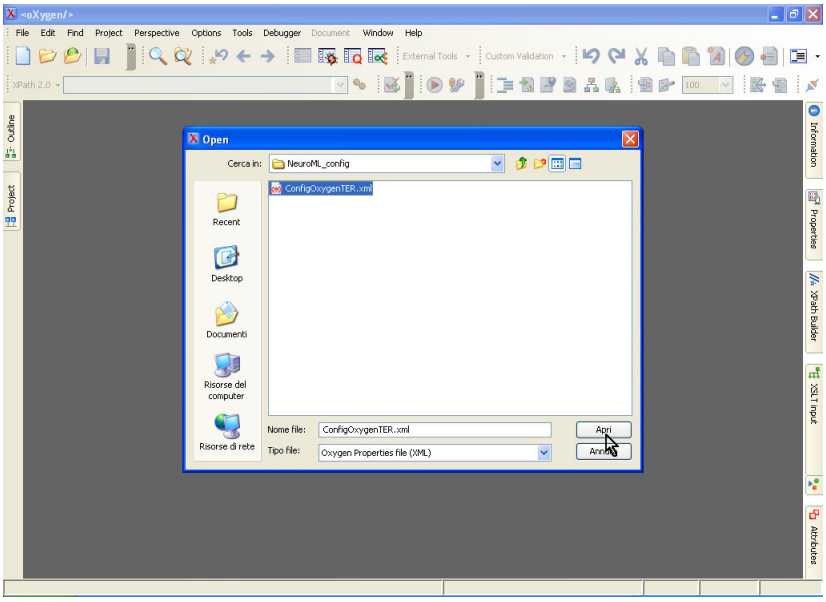
The model window gives an overview of the selected element and provides information on any attributes used by the element. The outline window shows the document in its tree format. The program can convert an existing XML document into a Schema or DTD.

We've configured the <oXygen/> editor to be easily used by the NeuroML Community. On top of that, we have integrated RELAX NG grammars written by us as an alternative to the NeuroML Schemata v1.1. The following instructions will allow you to incorporate the NeuroML envirement in the <oXygen/> software :

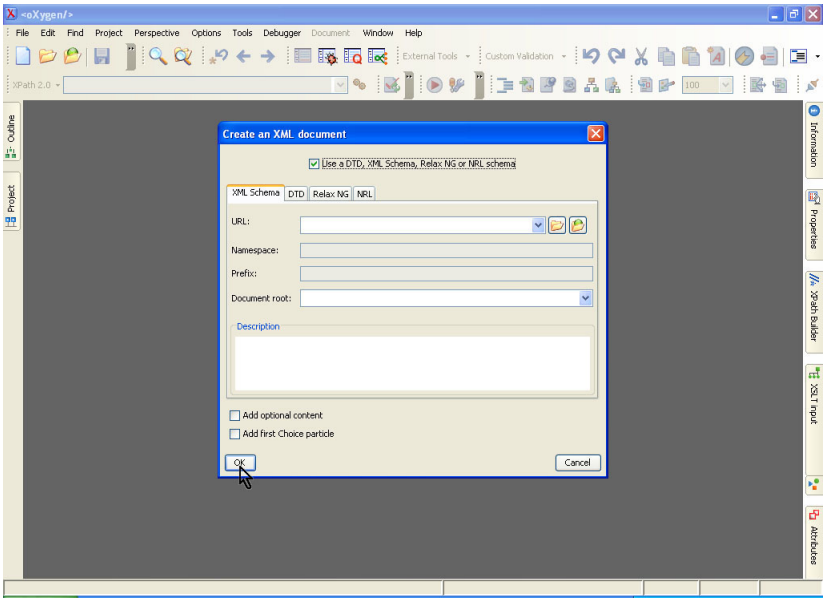
1. unzip the archive
2. copy the frameworks directory in the <oXygen/> directory (eventually overwrite)
3. run the <oXygen/> software
4. select the option menu and choose the import option



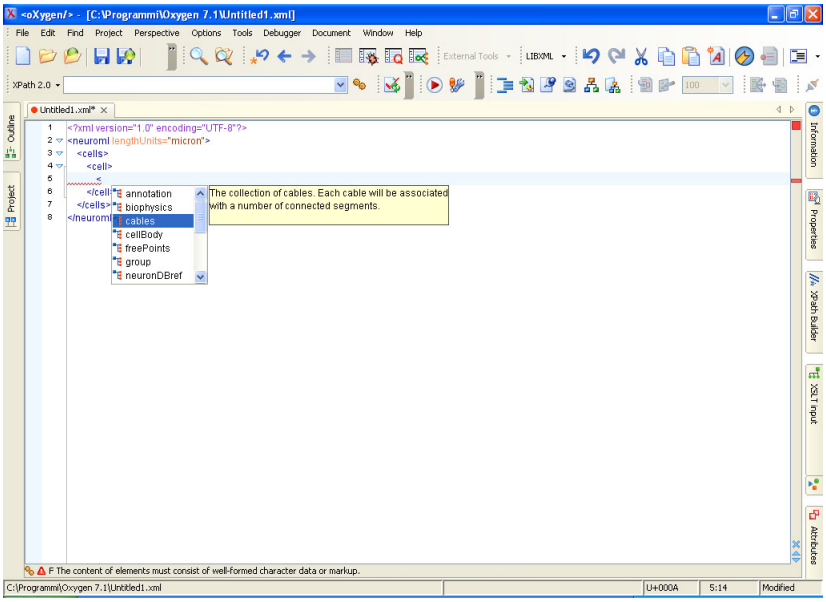
5. open the ConfigOxygenTER.xml file and confirm it



6. now you can create or edit an XML file. No additional option is required : you only need to confirm without adding any Schema to your file.



7. wait a few seconds until the analysis is complete
8. now you can start working


<oXygen/> - [C:\Programmi\Oxygen 7.1\Untitled1.xml]
File Edit Find Project Perspective Options Tools Debugger Document Window Help
Untitled1.xml*
<?xml version="1.0" encoding="UTF-8"?>
<neuroml lengthUnits="micron">
<cells>
<cell>
annotation
biophysics
cables
cellBody
freePoints
group
neuronDBref
The collection of cables. Each cable will be associated with a number of connected segments.
The content of elements must consist of well-formed character data or markup.
Modified

Java code source

A class XMLOut

```
package neuroml.util;

import neuroml.core.*;
import java.lang.reflect.*;
import java.util.Vector;
import java.util.Collection;
import java.util.Map;
import java.util.Iterator;
import java.io.*;

import org.jdom.*;
import org.jdom.output.XMLOutputter;
/**
 * Write out XML given an object - SBML style
 */
abstract public class XMLOut {
        /*********************************************************
         * *******************************************************
         * ***************** NEW CODE ******************
         * *******************************************************
         * *******************************************************
         */

        public static void XMLOutPut(NamedObject obj, Class cl, String fileName) {
                XMLOutputter outPutter = new XMLOutputter();
                Writer out = null;
                Element root = new Element("neuroml");
                Document doc = new Document(root);
                root.addAttribute("class", cl.getName());
                root.addAttribute("name", obj.getName());
                root = obj.toStringXML(root);
                try {
                        out = new FileWriter(fileName);
                        outPutter.output(doc, out);
                }
                catch (Exception ex) {
                        ex.printStackTrace();
                }
                finally {
                        try { out.close(); }
                        catch(Exception ex) { ex.printStackTrace(); }
```

```
                }
        }
}
```

B class CellPkg

```
package neuroml.model.cell;

import org.jdom.Element;

import neuroml.core.*;

import neuroml.model.channel.*;

/** Top Level Class for a collection of cells
 */
// add toStringXML() and toStringXML(Element root)
public class CellPkg extends Pkg {
        // modified accessibility of attributes
        private Set solutions = null; //new Set("Solution");
        private Set sectionProperties = null; //new Set("SectionProperties");
        private Set neurons = null; //new Set("AbstractNeuron");
        private Set structures = null; //new Set("NeuronStructure");

        // Added getter and setter for attributes
        public Set getSolutions() { return solutions; }
        public void setSolutions(Set sol) { this.solutions = sol; }
        public void addSolution(Solution sol) { this.solutions.add(sol); }

        public Set getSectionProperties() { return sectionProperties; }
        public void setSectionProperties(Set sp) { this.sectionProperties = sp; }
        public void addSectionProperties(SectionProperties sp) {
                this.sectionProperties.add(sp);
        }

        public Set getNeurons() { return neurons; }
        public void setNeurons(Set neurons) { this.neurons = neurons; }
        public void addNeuron(AbstractNeuron an) { this.neurons.add(an); }

        public Set getStructures() { return structures; }
        public void setStructures(Set structures) { this.structures = structures; }
        public void addStructure(NeuronStructure ns) { this.structures.add(ns); }

        // attribute initialisation must be in the constructor
        public CellPkg() {
                solutions = new Set("Solution");
```

```
        sectionProperties = new Set("SectionProperties");
        neurons = new Set("AbstractNeuron");
        structures = new Set("NeuronStructure");
}

// attribute initialisation must be in the constructor
public CellPkg(String name) {
        super(name);
        solutions = new Set("Solution");
        sectionProperties = new Set("SectionProperties");
        neurons = new Set("AbstractNeuron");
        structures = new Set("NeuronStructure");
}

/** A list of classes defined here
 * Used by the Namespace
 * @see neuroml.util.Namespace
 */
public static String[] getClassList() {
        String[] ret = {
                        "AbstractNeuron",
                        "CableNeuron",
                        "NeuronModule",
                        "ModuleNeuron",
                        "CellPkg",
                        "ChannelAssignment",
                        "Mask",
                        "NeuronStructure",
                        "OrientedSubstructure",
                        "PointNeuron",
                        "SectionProperties",
                        "SectionSet"
        };
        return ret;
}

// added toString methods
public Element toStringXML(Element root) {
        Element sols = new Element("solutions");
        if (solutions != null && solutions.size() != 0) {
                sols = solutions.toStringXML(sols);
        }

        Element sp = new Element("sectionProperties");
if (sectionProperties != null && sectionProperties.size() != 0) {
        sp = sectionProperties.toStringXML(sp);
```

```
        }

        Element ns = new Element("neurones");
        if (neurons != null && neurons.size() != 0) {
                ns = neurons.toStringXML(ns);
        }

        Element strs = new Element ("structures");
        if (structures != null && structures.size() != 0) {
                strs = structures.toStringXML(strs);
        }
        root.addContent(sols);
        root.addContent(sp);
        root.addContent(ns);
        root.addContent(strs);
                return root;
        }

}
```

C class Set

```
package neuroml.core;
import java.util.Vector;
import java.util.Collection;
import java.util.Iterator;
import org.jdom.Element;

/**
 * Core support for collections of objects.
 * This should be changed to java.util.Collection
 * so user classes don't need a dependency on this one.
 *
 * This one could become a "collection with default type"
 * and implement java.util.Collection
 */

public class Set extends NamedObject implements ContentRestricted,
        java.util.Collection  {

        // The parameters should be declared as private for security reasons
        //  accesors as well as setters will be added
        private String type;
        private Vector elements; // to become Set in future

        /** Construct an empty Set
```

```
 */

// type accessor and setter
public String getType(){ return type; }
public void setType(String type){ this.type = type; }

// added accessor and setter for elements attribute
public Vector getElements() { return elements; }
public void setElements(Vector vect) {this.elements = vect; }

// Changed the mother class name parameter way of initialisation :
//  name = new String() <=> super(new String());
public Set() {
        super(new String());
        elements=new Vector();
        type = new String();
}

/** Construct an empty Set of a default class
 */

// Changed the mother class name parameter way of initialisation :
//  name = new String() <=> super(new String());
public Set(String type) {
        super(new String());
        this.type = type;
        elements=new Vector();
}

// Implementation of java.util.Collection - delegated to elements.
public boolean add(Object o) { return elements.add(o); }
public boolean addAll(Collection c) { return elements.addAll(c); }

public void clear() { elements.clear(); }

public boolean contains(Object o) { return elements.contains(o); }
public boolean containsAll(Collection c) { return elements.containsAll(c); }

public boolean equals(Object o) { return elements.equals(o); }

public int hashCode() { return elements.hashCode(); }

public boolean isEmpty() { return elements.isEmpty(); }


public Iterator iterator() { return elements.iterator(); }
```

```
    public boolean remove(Object o) { return elements.remove(o); }
    public boolean removeAll(Collection c) { return elements.removeAll(c); }

    public boolean retainAll(Collection c) { return elements.retainAll(c); }

    public int size() { return elements.size(); }

    public Object[] toArray() {return elements.toArray(); }
    public Object[] toArray(Object[] a) { return elements.toArray(a); }

    // Implementation of ContentRestricted
    public String getContentClassName() { return type; }
    public void setContentClassName(String s) { type = s; }


    /** @deprecated use add() instead */
    public void addElement( Object o ) { elements.addElement(o); }
    /** @deprecated use remove() instead */
    public void removeElement( Object o ) { elements.removeElement(o); }
    /** @deprecated use getContentClassName() instead */
    public String getElementType() { return type; }

    // added toString methods
    public String toString() {
    String ret = "";
    Iterator it = iterator();
    NamedObject sub = null;
        while (it.hasNext()) {
            sub = (NamedObject) it.next();
            ret += sub.toString();
        }
        return ret;
}

public String toString(String nameTag) {
    String ret = "";
    Iterator it = iterator();
    NamedObject sub = null;
        while (it.hasNext()) {
            sub = (NamedObject) it.next();
            ret += "<"+ nameTag+"␣"+ sub.toString() + ">\n";
                    ret += "</"+ nameTag+">\n";
        }
        return ret;
}
```

```
    public Element toStringXML() {
        Element ret = null;
        Iterator it = iterator();
        NamedObject sub = null;
            while (it.hasNext()) {
                sub = (NamedObject) it.next();
                ret.addContent(sub.toStringXML());
            }
        return ret;
    }

    public Element toStringXML(Element root) {
        Iterator it = iterator();
        NamedObject sub = null;
            while (it.hasNext()) {
                sub = (NamedObject) it.next();
                root.addContent(sub.toStringXML());
            }
        return root;
    }
}
```

XML source class

D MultipleCells2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<neuroml name="Multiple cells" lengthUnits="micron">

    <cells>

        <cell name="Cell 1">
            <cellBody>
                <polygon>
                    <point x="10" y="20" z="30"/>
                    <point x="1" y="2" z="3"/>
                    <point x="11" y="22" z="33"/>
                    <point x="7" y="14" z="21"/>
                    <point x="5" y="10" z="15"/>
                </polygon>
            </cellBody>
            <biophysics> Some biophysical properties... </biophysics>
        </cell>
```

```
<cell name="MossyCell">
    <notes>An abstracted Mossy Cell morphology from Santhakumar
        et al 2005</notes>
    <segments>
        <segment id="0" cable="100">
            <proximal x="0.0" y="0.0" z="0.0" diameter="15.0"/>
            <distal x="0.0" y="20.0" z="0.0" diameter="15.0"/>
        </segment>
        <segment id="1" parent="0" cable="101">
            <proximal x="0.0" y="20.0" z="0.0" diameter="4.0"/>
            <distal x="29.206377" y="89.079575" z="0.0"
                diameter="4.0"/>
        </segment>
        <segment id="2" parent="1" cable="102">
            <proximal x="29.206377" y="89.079575" z="0.0"
                diameter="3.0"/>
            <distal x="58.412754" y="158.15915" z="0.0"
                diameter="3.0"/>
        </segment>
        <segment id="3" parent="2" cable="103">
            <proximal x="58.412754" y="158.15915" z="0.0"
                diameter="2.0"/>
            <distal x="87.61913" y="227.23872" z="0.0"
                diameter="2.0"/>
        </segment>
        <segment id="4" parent="3" cable="104">
            <proximal x="87.61913" y="227.23872" z="0.0"
                diameter="1.0"/>
            <distal x="116.825516" y="296.3183" z="0.0"
                diameter="1.0"/>
        </segment>
        <segment id="5" parent="0" cable="105">
            <proximal x="0.0" y="20.0" z="0.0" diameter="4.0"/>
            <distal x="-29.206377" y="89.079575" z="0.0"
                diameter="4.0"/>
        </segment>
        <segment id="6" parent="5" cable="106">
            <proximal x="-29.206377" y="89.079575" z="0.0"
                diameter="3.0"/>
            <distal x="-58.412754" y="158.15915" z="0.0"
                diameter="3.0"/>
        </segment>
```

```
<segment id="7" parent="6" cable="107">
    <proximal x="-58.412754" y="158.15915" z="0.0"
              diameter="2.0"/>
    <distal x="-87.61913" y="227.23872" z="0.0"
            diameter="2.0"/>
</segment>
<segment id="8" parent="7" cable="108">
    <proximal x="-87.61913" y="227.23872" z="0.0"
              diameter="1.0"/>
    <distal x="-116.825516" y="296.3183" z="0.0"
            diameter="1.0"/>
</segment>
<segment id="9" parent="0" cable="109">
    <proximal x="0.0" y="0.0" z="0.0" diameter="4.0"/>
    <distal x="-19.470915" y="-46.05305" z="0.0"
            diameter="4.0"/>
</segment>
<segment id="10" parent="9" cable="110">
    <proximal x="-19.470915" y="-46.05305" z="0.0"
              diameter="3.0"/>
    <distal x="-38.94183" y="-92.1061" z="0.0"
            diameter="3.0"/>
</segment>
<segment id="11" parent="10" cable="111">
    <proximal x="-38.94183" y="-92.1061" z="0.0"
              diameter="2.0"/>
    <distal x="-58.412743" y="-138.15915" z="0.0"
            diameter="2.0"/>
</segment>
<segment id="12" parent="11" cable="112">
    <proximal x="-58.412743" y="-138.15915" z="0.0"
              diameter="1.0"/>
    <distal x="-77.88365" y="-184.21219" z="0.0"
            diameter="1.0"/>
</segment>
<segment id="13" parent="0" cable="113">
    <proximal x="0.0" y="0.0" z="0.0" diameter="4.0"/>
    <distal x="19.470915" y="-46.05305" z="0.0"
            diameter="4.0"/>
</segment>
<segment id="14" parent="13" cable="114">
    <proximal x="19.470915" y="-46.05305" z="0.0"
```

```
                    diameter="3.0"/>
            <distal x="38.94183" y="-92.1061" z="0.0"
                    diameter="3.0"/>
        </segment>
        <segment id="15" parent="14" cable="115">
            <proximal x="38.94183" y="-92.1061" z="0.0"
                    diameter="2.0"/>
            <distal x="58.412743" y="-138.15915" z="0.0"
                    diameter="2.0"/>
        </segment>
        <segment id="16" parent="15" cable="116">
            <proximal x="58.412743" y="-138.15915" z="0.0"
                    diameter="1.0"/>
            <distal x="77.88365" y="-184.21219" z="0.0"
                    diameter="1.0"/>
        </segment>
    </segments>
    <cables>
        <cable id="100" name="soma">
            <group>all</group>
            <group>soma_group</group>
        </cable>
        <cable id="101" name="bcdend1_0">
            <group>all</group>
            <group>dendrite_group</group>
            <group>adend</group>
        </cable>
        <cable id="102" name="bcdend1_1">
            <group>all</group>
            <group>dendrite_group</group>
            <group>bdend</group>
            <group>bdend_apical</group>
        </cable>
        <cable id="103" name="bcdend1_2">
            <group>all</group>
            <group>dendrite_group</group>
            <group>cdend</group>
        </cable>
        <cable id="104" name="bcdend1_3">
            <group>all</group>
            <group>dendrite_group</group>
            <group>ddend</group>
```

```
        <group>ddend_apical</group>
    </cable>
    <cable id="105" name="bcdend2_0">
        <group>all</group>
        <group>dendrite_group</group>
        <group>adend</group>
    </cable>
    <cable id="106" name="bcdend2_1">
        <group>all</group>
        <group>dendrite_group</group>
        <group>bdend</group>
        <group>bdend_apical</group>
    </cable>
    <cable id="107" name="bcdend2_2">
        <group>all</group>
        <group>dendrite_group</group>
        <group>cdend</group>
    </cable>
    <cable id="108" name="bcdend2_3">
        <group>all</group>
        <group>dendrite_group</group>
        <group>ddend</group>
        <group>ddend_apical</group>
    </cable>
    <cable id="109" name="bcdend3_0">
        <group>all</group>
        <group>dendrite_group</group>
        <group>adend</group>
    </cable>
    <cable id="110" name="bcdend3_1">
        <group>all</group>
        <group>dendrite_group</group>
        <group>bdend</group>
    </cable>
    <cable id="111" name="bcdend3_2">
        <group>all</group>
        <group>dendrite_group</group>
        <group>cdend</group>
    </cable>
    <cable id="112" name="bcdend3_3">
        <group>all</group>
        <group>dendrite_group</group>
```

```
                <group>ddend</group>
            </cable>
            <cable id="113" name="bcdend4_0">
                <group>all</group>
                <group>dendrite_group</group>
                <group>adend</group>
            </cable>
            <cable id="114" name="bcdend4_1">
                <group>all</group>
                <group>dendrite_group</group>
                <group>bdend</group>
            </cable>
            <cable id="115" name="bcdend4_2">
                <group>all</group>
                <group>dendrite_group</group>
                <group>cdend</group>
            </cable>
            <cable id="116" name="bcdend4_3">
                <group>all</group>
                <group>dendrite_group</group>
                <group>ddend</group>
            </cable>
        </cables>
        <biophysics> Some biophysical properties... </biophysics>
    </cell>

    <cell name="Cell_3">
        <cellBody>
            <polygon>
                <point x="2" y="4" z="6"/>
                <point x="3" y="6" z="9"/>
                <point x="4" y="8" z="12"/>
                <point x="12" y="8" z="4"/>
                <point x="9" y="6" z="3"/>
                <point x="6" y="4" z="2"/>
            </polygon>
        </cellBody>
        <biophysics> Some biophysical properties... </biophysics>
    </cell>

</cells>
```

```
</neuroml>
```

E res.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<neuroml name="Unique_cell" lengthUnits="micron">
    <cells>
        <cell name="MossyCell">
            <segments>
                <segment id="0" cable="100" name="Segment_0">
                    <proximal diameter="15.0" x="0.0" y="0.0" z="0.0"/>
                    <distal diameter="15.0" x="0.0" y="20.0" z="0.0"/>
                </segment>
                <segment id="1" cable="101" name="Segment_1" parent="0">
                    <proximal diameter="4.0" x="0.0" y="20.0" z="0.0"/>
                    <distal diameter="4.0" x="29.206377" y="89.079575"
                            z="0.0"/>
                </segment>
                <segment id="2" cable="102" name="Segment_2" parent="1">
                    <proximal diameter="3.0" x="29.206377" y="89.079575"
                              z="0.0"/>
                    <distal diameter="3.0" x="58.412754" y="158.15915"
                            z="0.0"/>
                </segment>
                <segment id="3" cable="103" name="Segment_3" parent="2">
                    <proximal diameter="2.0" x="58.412754" y="158.15915"
                              z="0.0"/>
                    <distal diameter="2.0" x="87.61913" y="227.23872"
                            z="0.0"/>
                </segment>
                <segment id="4" cable="104" name="Segment_4" parent="3">
                    <proximal diameter="1.0" x="87.61913" y="227.23872"
                              z="0.0"/>
                    <distal diameter="1.0" x="116.825516" y="296.3183"
                            z="0.0"/>
                </segment>
                <segment id="5" cable="105" name="Segment_5" parent="0">
                    <proximal diameter="4.0" x="0.0" y="20.0" z="0.0"/>
                    <distal diameter="4.0" x="-29.206377" y="89.079575"
                            z="0.0"/>
                </segment>
                <segment id="6" cable="106" name="Segment_6" parent="5">
```

```
    <proximal diameter="3.0" x="-29.206377" y="89.079575"
              z="0.0"/>
    <distal diameter="3.0" x="-58.412754" y="158.15915"
            z="0.0"/>
</segment>
<segment id="7" cable="107" name="Segment_7" parent="6">
    <proximal diameter="2.0" x="-58.412754" y="158.15915"
              z="0.0"/>
    <distal diameter="2.0" x="-87.61913" y="227.23872"
            z="0.0"/>
</segment>
<segment id="8" cable="108" name="Segment_8" parent="7">
    <proximal diameter="1.0" x="-87.61913" y="227.23872"
              z="0.0"/>
    <distal diameter="1.0" x="-116.825516" y="296.3183"
            z="0.0"/>
</segment>
<segment id="9" cable="109" name="Segment_9" parent="0">
    <proximal diameter="4.0" x="0.0" y="0.0" z="0.0"/>
    <distal diameter="4.0" x="-19.470915" y="-46.05305"
            z="0.0"/>
</segment>
<segment id="10" cable="110" name="Segment_10" parent="9">
    <proximal diameter="3.0" x="-19.470915" y="-46.05305"
              z="0.0"/>
    <distal diameter="3.0" x="-38.94183" y="-92.1061"
            z="0.0"/>
</segment>
<segment id="11" cable="111" name="Segment_11" parent="10">
    <proximal diameter="2.0" x="-38.94183" y="-92.1061"
              z="0.0"/>
    <distal diameter="2.0" x="-58.412743" y="-138.15915"
            z="0.0"/>
</segment>
<segment id="12" cable="112" name="Segment_12" parent="11">
    <proximal diameter="1.0" x="-58.412743" y="-138.15915"
              z="0.0"/>
    <distal diameter="1.0" x="-77.88365" y="-184.21219"
            z="0.0"/>
</segment>
<segment id="13" cable="113" name="Segment_13" parent="0">
    <proximal diameter="4.0" x="0.0" y="0.0" z="0.0"/>
```

```
            <distal diameter="4.0" x="19.470915" y="-46.05305"
                z="0.0"/>
        </segment>
        <segment id="14" cable="114" name="Segment_14" parent="13">
            <proximal diameter="3.0" x="19.470915" y="-46.05305"
                z="0.0"/>
            <distal diameter="3.0" x="38.94183" y="-92.1061"
                z="0.0"/>
        </segment>
        <segment id="15" cable="115" name="Segment_15" parent="14">
            <proximal diameter="2.0" x="38.94183" y="-92.1061"
                z="0.0"/>
            <distal diameter="2.0" x="58.412743" y="-138.15915"
                z="0.0"/>
        </segment>
        <segment id="16" cable="116" name="Segment_16" parent="15">
            <proximal diameter="1.0" x="58.412743" y="-138.15915"
                z="0.0"/>
            <distal diameter="1.0" x="77.88365" y="-184.21219"
                z="0.0"/>
        </segment>
    </segments>
    <cables>
        <cable id="100" name="soma">
            <group>all</group>
            <group>soma_group</group>
        </cable>
        <cable id="101" name="bcdend1_0">
            <group>all</group>
            <group>dendrite_group</group>
            <group>adend</group>
        </cable>
        <cable id="102" name="bcdend1_1">
            <group>all</group>
            <group>dendrite_group</group>
            <group>bdend</group>
            <group>bdend_apical</group>
        </cable>
        <cable id="103" name="bcdend1_2">
            <group>all</group>
            <group>dendrite_group</group>
            <group>cdend</group>
```

```
</cable>
<cable id="104" name="bcdend1_3">
    <group>all</group>
    <group>dendrite_group</group>
    <group>ddend</group>
    <group>ddend_apical</group>
</cable>
<cable id="105" name="bcdend2_0">
    <group>all</group>
    <group>dendrite_group</group>
    <group>adend</group>
</cable>
<cable id="106" name="bcdend2_1">
    <group>all</group>
    <group>dendrite_group</group>
    <group>bdend</group>
    <group>bdend_apical</group>
</cable>
<cable id="107" name="bcdend2_2">
    <group>all</group>
    <group>dendrite_group</group>
    <group>cdend</group>
</cable>
<cable id="108" name="bcdend2_3">
    <group>all</group>
    <group>dendrite_group</group>
    <group>ddend</group>
    <group>ddend_apical</group>
</cable>
<cable id="109" name="bcdend3_0">
    <group>all</group>
    <group>dendrite_group</group>
    <group>adend</group>
</cable>
<cable id="110" name="bcdend3_1">
    <group>all</group>
    <group>dendrite_group</group>
    <group>bdend</group>
</cable>
<cable id="111" name="bcdend3_2">
    <group>all</group>
    <group>dendrite_group</group>
```

```
                    <group>cdend</group>
                </cable>
                <cable id="112" name="bcdend3_3">
                    <group>all</group>
                    <group>dendrite_group</group>
                    <group>ddend</group>
                </cable>
                <cable id="113" name="bcdend4_0">
                    <group>all</group>
                    <group>dendrite_group</group>
                    <group>adend</group>
                </cable>
                <cable id="114" name="bcdend4_1">
                    <group>all</group>
                    <group>dendrite_group</group>
                    <group>bdend</group>
                </cable>
                <cable id="115" name="bcdend4_2">
                    <group>all</group>
                    <group>dendrite_group</group>
                    <group>cdend</group>
                </cable>
                <cable id="116" name="bcdend4_3">
                    <group>all</group>
                    <group>dendrite_group</group>
                    <group>ddend</group>
                </cable>
            </cables>
        </cell>
    </cells>
</neuroml>
```

RELAX NG source code

F NeuroML_v1.1.rng

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

<!-- This RELAX NG schema could be an alternative to XML schemas -->

<!-- It should link MorphML from Level 1 with Biophysics (specifying
     placement of channels on cells) and ChannelML (specifying the
     channel mechanism) -->
```

```
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
    datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

    <include href="./Level1/Metadata_v1.1.rng"/>
    <include href="./Level1/MorphML_v1.1.rng"/>
    <include href="./Level2/Biophysics_v1.1.rng"/>
    <include href="./Level2/ChannelML_v1.1.rng"/>

    <start>
        <choice>
            <element name="neuroml">
                <ref name="NeuroML"/>
            </element>
            <element name="morphml">
                <ref name="Morphology"/>
            </element>
            <element name="channelml">
                <ref name="ChannelML"/>
            </element>
        </choice>
    </start>

    <define name="NeuroML">
        <a:documentation>Description of neuronal models, including
            biophysics and channel mechanisms.</a:documentation>
        <ref name="metadata"/>
        <ref name="referencedata"/>
        <element name="cells">
            <ref name="Cells"/>
        </element>
        <optional>
            <element name="channels">
                <ref name="Channels"/>
            </element>
        </optional>
        <optional>
            <element name="propertyDetails">
                <ref name="PropertyDetails"/>
            </element>
        </optional>
        <optional>
```

```
            <element name="groupDetails">
                <ref name="GroupDetails"/>
            </element>
        </optional>
        <ref name="NameAttr"/>
        <attribute name="lengthUnits">
            <a:documentation>
                Unit of all length measurements.
            </a:documentation>
            <ref name="LengthUnits"/>
        </attribute>
        <optional>
            <attribute name="volumeUnits">
                <a:documentation>
                    Unit of all volume measurements.
                </a:documentation>
                <ref name="VolumeUnits"/>
                <!-- QUESTION how to set a default value for
                    this attribute -->
            </attribute>
        </optional>
    </define>

    <define name="Cells">
        <a:documentation>Collection of all cells.</a:documentation>
        <oneOrMore>
            <element name="cell">
                <a:documentation>A single cell specified in MorphML
                    extended to include channel density info.
                </a:documentation>
                <ref name="BioCell"/>
            </element>
        </oneOrMore>
    </define>

    <define name="BioCell">
        <a:documentation>
            Cell with extensions for biophysics.
        </a:documentation>
        <ref name="Cell"/>
        <element name="biophysics">
            <ref name="Biophysics"/>
```

```
        </element>
    </define>

    <define name="Channels">
        <a:documentation>Collection of all channels.</a:documentation>
        <zeroOrMore>
            <element name="ion">
                <ref name="Ion"/>
            </element>
        </zeroOrMore>
        <optional>
            <ref name="ChannelType"/>
        </optional>
        <optional>
            <ref name="SynapseType"/>
        </optional>
        <optional>
            <ref name="IonConcentration"/>
        </optional>
    </define>

</grammar>
```

G Metadata_v1.1.rng

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

  <define name="NameAttr" combine="choice">
    <!-- name attribute -->
    <optional>
      <attribute name="name">
        <data type="string"/>
      </attribute>
    </optional>
  </define>

  <define name="Point" combine="choice">
    <a:documentation>
      A 3D point with optional diameter.
```

```
    </a:documentation>
    <attribute name="x">
      <data type="double"/>
    </attribute>
    <attribute name="y">
      <data type="double"/>
    </attribute>
    <attribute name="z">
      <data type="double"/>
    </attribute>
    <optional>
      <attribute name="diameter">
        <data type="double"/>
      </attribute>
    </optional>
  </define>

  <define name="Points" combine="choice">
    <a:documentation>A collection of points.</a:documentation>
    <oneOrMore>
      <element name="point">
        <ref name="Point"/>
      </element>
    </oneOrMore>
    <ref name="NameAttr"/>
  </define>

  <define name="Sphere" combine="choice">
    <a:documentation>
      A spherical structure such as a cell body or cell.
    </a:documentation>
    <element name="center">
      <ref name="Point"/>
    </element>
    <ref name="NameAttr"/>
  </define>

  <define name="Manifold" combine="choice">
    <a:documentation>A surface.</a:documentation>
    <ref name="Points"/>
  </define>
```

```
<define name="Polygon" combine="choice">
  <a:documentation>A closed structure represented by a list of
    points where the first point connects with the last point.
  </a:documentation>
  <!-- IDEA we can express a data constraint for the connection
    of the first point avec the last pont -->
  <ref name="Points"/>
</define>

<define name="Polyhedron" combine="choice">
  <a:documentation>A 3d surface to represent the cell body or
    histological structure.</a:documentation>
  <element name="polygons">
    <a:documentation>Collection of polygons defining the polyhedron.
    </a:documentation>
    <oneOrMore>
      <element name="polygon">
        <ref name="Polygon"/>
      </element>
    </oneOrMore>
  </element>
</define>

<define name="Annotation" combine="choice">
  <a:documentation>
    Concise processing directives for downstream applications.
  </a:documentation>
  <!-- TODO -->
  <text/>
</define>

<define name="LengthUnits" combine="choice">
  <a:documentation>Enumeration of length units. Used in MorphML
    Level 1 files , where length is the only important dimension.
  </a:documentation>
  <choice>
    <value>micron</value>
    <value>millimetre</value>
    <value>metre</value>
  </choice>
</define>
```

```
<define name="VolumeUnits" combine="choice">
  <a:documentation>Enumeration of volume units.</a:documentation>
  <choice>
    <value>cubic_millimetre</value>
    <value>millilitre</value>
    <value>litre</value>
  </choice>
</define>

<define name="Notes" combine="choice">
  <a:documentation>Textual human readable notes related to the
    element in question</a:documentation>
  <data type="string"/>
</define>

<define name="Group" combine="choice">
  <a:documentation>Allows elements to be associated, such as for
    grouping segments or cables into
    the basal arbor.</a:documentation>
  <!-- QUESTION what is it -->
  <data type="string"/>
</define>

<define name="Property" combine="choice">
  <a:documentation>A Tag/Value/Type tuple.</a:documentation>
  <!-- QUESTION wher is Type QUESTION should Tag Value Type
    always be present -->
  <interleave>
    <element name="tag">
      <data type="string"/>
    </element>
    <element name="value">
      <data type="string"/>
    </element>
  </interleave>
</define>

<define name="Properties" combine="choice">
  <a:documentation>A collection of Properties</a:documentation>
  <zeroOrMore>
    <element name="property">
      <ref name="Property"/>
```

```
      </element>
    </zeroOrMore>
  </define>

  <define name="GroupDetail" combine="choice">
    <a:documentation>Metadata for each Group.</a:documentation>
    <element name="description">
      <data type="string"/>
    </element>
    <zeroOrMore>
      <element name="properties">
        <!-- QUESTION why do we need a set of a set of properties -->
        <ref name="Properties"/>
      </element>
    </zeroOrMore>
    <attribute name="group">
      <data type="string"/>
    </attribute>
  </define>

  <define name="YesNo" combine="choice">
    <a:documentation>String with only yes or no allowed.
    </a:documentation>
    <choice>
      <value>yes</value>
      <value>no</value>
    </choice>
  </define>

  <define name="PropertyDetail" combine="choice">
    <a:documentation>Metadata for each Property.</a:documentation>
    <interleave>
      <element name="description">
        <data type="string"/>
      </element>
      <element name="type">
        <data type="string"/>
        <!-- TODO anyType -->
      </element>
    </interleave>
    <attribute name="property">
      <data type="string"/>
```

```
    </attribute>
  </define>

  <define name="Publication" combine="choice">
    <a:documentation>A reference to a publication.</a:documentation>
    <interleave>
      <element name="fullTitle">
        <data type="string"/>
      </element>
      <element name="pubmedRef">
        <data type="string"/>
        <!-- PAY ATTENTION CAUSE NOT GOOD -->
      </element>
    </interleave>
  </define>

  <define name="NeuronDBReference" combine="choice">
    <a:documentation>A reference to an entity in NeuronDB.
    </a:documentation>
    <interleave>
      <element name="modelName">
        <data type="string"/>
      </element>
      <element name="uri">
        <data type="string"/>
        <!-- PAY ATTENTION CAUSE NOT GOOD -->
      </element>
    </interleave>
  </define>

  <define name="metadata" combine="choice">
    <a:documentation>
      General metadata which can be applied to a number of elements.
    </a:documentation>
    <optional>
      <element name="notes">
        <ref name="Notes"/>
      </element>
    </optional>
    <optional>
      <element name="properties">
        <ref name="Properties"/>
```

```
      </element>
    </optional>
    <optional>
      <element name="annotation">
        <ref name="Annotation"/>
      </element>
    </optional>
    <zeroOrMore>
      <element name="group">
        <ref name="Group"/>
      </element>
    </zeroOrMore>
  </define>

  <define name="referencedata" combine="choice">
    <a:documentation>
      General metadata which can be applied to a number of elements.
    </a:documentation>
    <optional>
      <element name="publication">
        <ref name="Publication"/>
      </element>
    </optional>
    <optional>
      <element name="neuronDBref">
        <ref name="NeuronDBReference"/>
      </element>
    </optional>
  </define>

  <define name="PropertyDetails" combine="choice">
    <a:documentation>
      Collection of all PropertyDetails for this instance.
    </a:documentation>
    <oneOrMore>
      <element name="propertyDetail">
        <ref name="PropertyDetail"/>
      </element>
    </oneOrMore>
    <!-- QUESTION how to represent xs:key keyProperty -->
    <!-- maybe by simply using ID IDREF from XML Schema datatype -->
    <!-- or maybe with some Schematron constraints -->
```

```
  </define>

  <define name="GroupDetails" combine="choice">
    <a:documentation>
      Collection of all GroupDetails for this instance.
    </a:documentation>
    <oneOrMore>
      <element name="groupDetail">
        <ref name="GroupDetail"/>
      </element>
    </oneOrMore>
    <!-- QUESTION how to represent xs:key keyGroup -->
  </define>

</grammar>
```

H MorphML_v1.1.rng

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

  <include href="./Metadata_v1.1.rng"/>

  <define name="Morphology">
    <a:documentation>The main element which details the neuronal
      morphology. Cells, various histological features, and
      properties associated with the data can be contained in this
      element.
    </a:documentation>

    <ref name="metadata"/>

    <element name="cells">
      <a:documentation>Collection of all cells.</a:documentation>
      <oneOrMore>
        <element name="cell">
          <ref name="Cell"/>
        </element>
      </oneOrMore>
    </element>
```

```
<optional>
  <element name="features">
    <a:documentation>
      Collection of all extracellular histological features.
    </a:documentation>
    <ref name="metadata"/>
    <oneOrMore>
      <element name="feature">
        <ref name="Feature"/>
        <!-- QUESTION how to represent xs:key keyPath -->
        <!-- QUESTION how to represent xs:keyref pathParentRef -->
        <!-- maybe by simply using ID IDREF from XML Schema datatype -->
        <!-- or maybe with some Schematron constraints -->
      </element>
    </oneOrMore>
  </element>
</optional>

<optional>
  <element name="propertyDetails">
    <ref name="PropertyDetails"/>
  </element>
</optional>

<optional>
  <element name="groupDetails">
    <ref name="GroupDetails"/>
  </element>
</optional>

<ref name="NameAttr"/>
<attribute name="lengthUnits">
  <a:documentation>Unit of all length measurements.
  </a:documentation>
  <ref name="LengthUnits"/>
</attribute>
<optional>
  <attribute name="volumeUnits">
    <a:documentation>Unit of all volume measurements.
    </a:documentation>
    <ref name="VolumeUnits"/>
```

```
        <!-- QUESTION how to set a default value for this
          attribute -->
      </attribute>
    </optional>

  </define>

  <define name="Cell">
    <a:documentation>Definition of a cell.</a:documentation>
    <ref name="metadata"/>
    <ref name="referencedata"/>
    <optional>
      <element name="cellBody">
        <a:documentation>Used for anatomical representation of the
          soma. Use a Segment with equivalent properties to retain
          connectivity of branches to the soma for downstream
          applications (e.g. neuronal simulators).</a:documentation>
        <ref name="metadata"/>
        <choice>
          <element name="polygon">
            <ref name="Polygon"/>
          </element>
          <element name="polyhedron">
            <ref name="Polyhedron"/>
          </element>
          <element name="sphere">
            <ref name="Sphere"/>
          </element>
        </choice>
      </element>
    </optional>
    <zeroOrMore>
      <element name="segments">
        <ref name="metadata"/>
        <oneOrMore>
          <element name="segment">
            <ref name="Segment"/>

            <sch:pattern name="Test_references"
              xmlns:sch="http://www.ascc.net/xml/schematron">
              <sch:rule context="segment">
                <sch:assert test="@id">
```

```
                An element of type segment should have an id attribute that
                is a unique identifier. </sch:assert>
            </sch:rule>
            <sch:rule context="segment[@parent]"
              role="referenced element must be of a valid type">
              <sch:assert test="(name(id(@parent))='segment')">
                The parent for a segment should be a segment.
              </sch:assert>
            </sch:rule>
            <sch:rule context="segment[@cable]"
              role="referenced element must be of a valid type">
              <sch:assert test="(name(id(@cable))='cable')">
                The referenced cable for a segment should be a cable.
              </sch:assert>
            </sch:rule>
          </sch:pattern>

        </element>
      </oneOrMore>
      <ref name="NameAttr"/>
      <!-- QUESTION how to represent xs:key keySegment -->
      <!-- QUESTION how to represent xs:keyref segmentParentRef -->
      <!-- maybe by simply using ID IDREF from XML Schema datatype -->
      <!-- or maybe with some Schematron constraints -->
    </element>
  </zeroOrMore>
  <optional>
    <element name="freePoints">
      <ref name="FreePoints"/>
    </element>
  </optional>
  <optional>
    <element name="spines">
      <ref name="metadata"/>
      <oneOrMore>
        <element name="spine">
          <ref name="Spine"/>
        </element>
      </oneOrMore>
      <!-- QUESTION how to represent xs:keyref spineParentRef -->
    </element>
  </optional>
```

```
    <optional>
      <element name="cables">
        <a:documentation>The collection of cables.
          Each cable will be associated with a number of
          connected segments.</a:documentation>
        <ref name="metadata"/>
        <oneOrMore>
          <element name="cable">
            <ref name="Cable"/>
          </element>
        </oneOrMore>
        <!-- QUESTION how to represent xs:key keyCable -->
        <!-- QUESTION how to represent xs:keyref cableParentRef -->
      </element>
    </optional>
    <ref name="NameAttr"/>
  </define>

  <define name="Segment">
    <a:documentation>Defines the smallest unit within a possibly
      branching structure, such as a dendrite or axon.
      The parent attribute is used to define connectivity.
      A segment would be mapped to a compartment in a compartmental
      modelling application such as GENESIS.</a:documentation>
    <optional>
      <element name="proximal">
        <a:documentation>The start point (and diameter) of the segment.
          If absent, it is assumed that the distal point of the
          parent is the start point of this segment.</a:documentation>
        <ref name="Point"/>
      </element>
    </optional>
    <element name="distal">
      <a:documentation>The end point (and diameter) of the segment.
      </a:documentation>
      <ref name="Point"/>
    </element>
    <optional>
      <element name="properties">
        <ref name="Properties"/>
      </element>
    </optional>
```

```
    <attribute name="id">
      <a:documentation>The ID of the segment, which should be unique
        within the cell.</a:documentation>
      <data type="nonNegativeInteger"/>
      <!-- ID -->
    </attribute>
    <ref name="NameAttr"/>
    <optional>
      <attribute name="parent">
        <a:documentation>Used to indicate logical connectivity
          between segments. Note that there is no requirement
          that the end point of this parent segment is equal to
          the start point of
          this segment.</a:documentation>
        <data type="nonNegativeInteger"/>
        <!-- IDREF -->
      </attribute>
    </optional>
    <optional>
      <attribute name="cable">
        <a:documentation>Used to indicate logical connectivity
          between segments. Note that there is no requirement that
          the end point of this parent segment is equal to the
          start point of this segment.</a:documentation>
        <data type="nonNegativeInteger"/>
        <!-- IDREF -->
      </attribute>
    </optional>
  </define>

  <define name="Cable">
    <a:documentation>Definition of a cable. A cable is a non-branching
      group of connected segments. A cable would be mapped to a
      section in a cable modelling based simulator such as NEURON.
    </a:documentation>
    <ref name="metadata"/>
    <attribute name="id">
      <a:documentation>ID of the cable, unique within this cell.
      </a:documentation>
      <data type="nonNegativeInteger"/>
      <!-- ID -->
    </attribute>
```

```
    <ref name="NameAttr"/>
    <optional>
      <attribute name="parent">
        <a:documentation>A cable ID. A way to connect cables logically.
        </a:documentation>
        <data type="nonNegativeInteger"/>
        <!-- IDREF -->
      </attribute>
    </optional>
    <optional>
      <attribute name="fractAlongParent">
        <a:documentation>
          Approximate location of this group s proximal point
          along the parent cable.
        </a:documentation>
        <data type="double"/>
      </attribute>
    </optional>
  </define>

  <define name="SpineShape">
    <a:documentation>Enumeration of allowed spine shapes.
    </a:documentation>
    <choice>
      <value>mushroom</value>
      <value>stubby</value>
      <value>thin</value>
    </choice>
  </define>

  <define name="Spine">
    <a:documentation>A spine with location, shape, and direction.
    </a:documentation>
    <element name="proximal">
      <ref name="Point"/>
    </element>
    <optional>
      <element name="distal">
        <ref name="Point"/>
      </element>
    </optional>
    <optional>
```

```
      <attribute name="parent">
        <a:documentation>The segment with which this spine is
          associated.</a:documentation>
        <data type="nonNegativeInteger"/>
        <!-- IDREF -->
      </attribute>
    </optional>
    <optional>
      <attribute name="length">
        <data type="double"/>
      </attribute>
    </optional>
    <optional>
      <attribute name="volume">
        <data type="double"/>
      </attribute>
    </optional>
    <optional>
      <attribute name="shape">
        <ref name="SpineShape"/>
      </attribute>
    </optional>
  </define>

  <define name="FreePoints">
    <a:documentation>
      The collection of varicosities or synaptic connections.
    </a:documentation>
    <ref name="Points"/>
  </define>

  <define name="Path">
    <a:documentation>Possibly branching histological structures.
    </a:documentation>
    <ref name="Points"/>
    <attribute name="id">
      <data type="nonNegativeInteger"/>
      <!-- ID -->
    </attribute>
    <optional>
      <attribute name="parent">
        <data type="nonNegativeInteger"/>
```

```
        <!-- IDREF -->
      </attribute>
    </optional>
  </define>

  <define name="Feature">
    <a:documentation>A single feature of note.</a:documentation>
    <ref name="metadata"/>
    <zeroOrMore>
      <element name="path">
        <ref name="Path"/>
      </element>
    </zeroOrMore>
    <zeroOrMore>
      <element name="freePoints">
        <ref name="FreePoints"/>
      </element>
    </zeroOrMore>
    <zeroOrMore>
      <element name="manifold">
        <ref name="Manifold"/>
      </element>
    </zeroOrMore>
    <zeroOrMore>
      <element name="polygon">
        <ref name="Polygon"/>
      </element>
    </zeroOrMore>
    <zeroOrMore>
      <element name="polyhedron">
        <ref name="Polyhedron"/>
      </element>
    </zeroOrMore>
    <zeroOrMore>
      <element name="sphere">
        <ref name="Sphere"/>
      </element>
    </zeroOrMore>
  </define>

</grammar>
```

I Biophysics_v1.1.rng

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

  <define name="Biophysics">
    <!-- TODO -->
    <text/>
  </define>

  <define name="Mechanism">
    <!-- TODO -->
    <text/>
  </define>

  <define name="MechanismType">
    <!-- TODO -->
    <text/>
  </define>

  <define name="BioUnits">
    <!-- since it is the same as ChaUnit of ChannelML a unique
      definition of Unit could be factorized in Metadata -->
    <!-- TODO -->
    <text/>
  </define>

</grammar>
```

J ChannelML_v1.1.rng

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

  <define name="ChannelML">
    <!-- TODO -->
    <text/>
  </define>
```

```
<define name="ChannelType">
  <!-- TODO -->
  <text/>
</define>

<define name="SynapseType">
  <!-- TODO -->
  <text/>
</define>

<define name="DoubleExponentialSynapse">
  <!-- TODO -->
  <text/>
</define>

<define name="CurrentVoltageRelation">
  <!-- TODO -->
  <text/>
</define>

<define name="Ohmic">
  <!-- TODO -->
  <text/>
</define>

<define name="RateAdjustments">
  <!-- TODO -->
  <text/>
</define>

<define name="Ion">
  <!-- TODO -->
  <text/>
</define>

<define name="IonRole">
  <!-- TODO -->
  <text/>
</define>

<define name="Gate">
```

```
    <!-- TODO -->
    <text/>
  </define>

  <define name="Transition">
    <!-- TODO -->
    <text/>
  </define>

  <define name="VoltageGate">
    <!-- TODO -->
    <text/>
  </define>

  <define name="VoltageConcGate">
    <!-- TODO -->
    <text/>
  </define>

  <define name="ConcDependence">
    <!-- TODO -->
    <text/>
  </define>

  <define name="AlphaBetaForm">
    <!-- TODO -->
    <text/>
  </define>

  <define name="AlphaBetaFormVoltConcDep">
    <!-- TODO -->
    <text/>
  </define>

  <define name="RateConstantEqnChoice">
    <!-- TODO -->
    <text/>
  </define>

  <define name="RateConstVoltConcDep">
    <!-- TODO -->
    <text/>
```

```
</define>

<define name="RateConstantEqn">
  <!-- TODO -->
  <text/>
</define>

<define name="AkdEquation">
  <!-- TODO -->
  <text/>
</define>

<define name="GenericEquation">
  <!-- TODO -->
  <text/>
</define>

<define name="IonConcentration">
  <!-- TODO -->
  <text/>
</define>

<define name="DecayingPoolModel">
  <!-- TODO -->
  <text/>
</define>

<define name="PoolVolumeInfo">
  <!-- TODO -->
  <text/>
</define>

<define name="Parameter">
  <!-- TODO -->
  <text/>
</define>

<define name="CoreEquationType">
  <!-- TODO -->
  <text/>
</define>
```

```
  <define name="ChaUnits">
    <!-- since it is the same as BioUnit a unique definition of
      unit could be factorized in Metadata -->
    <!-- TODO -->
    <text/>
  </define>

</grammar>
```

Principales définitions et acronymes

NeuroML
Neuron Markup Language. C'est une application XML permettant de décrire des modèles et des réseaux de neurones. Depuis sa version 1.1, NeuroML se constitue de plusieurs couches et il délègue une modélisation spécifique à chacune d'elles.

MorphML
C'est une application XML permettant des données sur la morphologie des neurones. Elle peut aussi être utilisée pour spécifier la structure de cellules dans l'environnement NeuroML et elle fait partie de sa couche de niveau 1.

ChannelML
Il s'agit d'une application XML similaire à MorphML, permettant de décrire les propriétés des canaux et les mécanismes synaptiques. Actuellement ChannelML fait partie de la couche de niveau 2 de NeuroML.

RelaxNG
REgular LAnguage for XML Next Generation. C'est un langage de description de document XML issu de la fusion de TreX de James Clark et de Relax de Murata Makoto. Considéré comme une alternative crédible de XML Schema, c'est un dialecte XML permettant de définir précisément les différentes contraintes qui déterminent la classe de documents XML pouvant passer l'étape de validation.

XML Schema
C'est un langage de description de format de document XML permettant d'en définir sa structure. Un schéma XML est lui-même un fichier XML.

Schematron
Un langage de spécification/validation de documents XML basé sur la définition de règles (avec XPath) et capable de reconnaître des motifs dans le document "parsé".

RDF
Resource Description Framework. C'est un modèle de graphes pour décrire les (méta-)données et d'en permettre un certain traitement automatique. Une des syntaxes (sérialisation) de ce langage est RDF/XML.

CDuce
C'est un langage de programmation fonctionnel, fortement typé, adapté à la manipulation sûre et efficace de documents XML.

Neurone impulsionnel
Ce terme est utilisé dans deux cadres différents : dans le cadre des travaux de modélisation en neurosciences : en particulier, on s'intéresse aux mécanismes physiologiques permettant l'apparition des impulsions, ainsi qu'au problème du codage neuronal , l'interprétation des échanges fonctionnels d'information. dans le cadre d'études informatiques sur les réseaux de neurones artificiels (connexionnisme). Ces neurones offrent la possibilité d'introduire naturellement une dimension temporelle dans des réseaux de neurones. Ils se représentent ainsi comme une forme particulière de système dynamique. Les neurones impulsionnels sont caractérisés par leurs puissances calculatoires (d'après certaines études), mais aussi utilisés dans le domaine de la robotique autonome. La simplicité du vecteur de communication utilisé offre un champ d'étude intéressant pour l'informatique (impulsions binaires), voire pour des applications électroniques.

References

http://www.neuroml.org/. The NeuroML Development Kit
http://morphml.org:8080/NeuroMLValidator/. NeuroML Validation service
http://www-sop.inria.fr/odyssee/contracts/rivage/about-neuroml/index.pdf. Using NeuroML
http://www.icsb2001.org/Posters/117_howell.pdf. NeuroML : A lingua franca for computational modeling in Neuroscience
http://home.earthlink.net/ perlewitz/sftwr.html. Computational neuroscience
http://www.brains-minds-media.org/archive/228/ XML for Model Specification in Neuroscience
http://www.anc.ed.ac.uk/ fwh/repository/nmlschematest/nmlschematest.html. Schema conversion: Java classes, .jar files and XML
http://www.w3.org/TR/REC-html40/sgml/dtd.html. DTD - HTML 4 Document Type Definition
http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/. XML Schema Part 0: Primer Second Edition
http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/. XML Schema Part 2: Primer Second Edition

http://www.oasis-open.org/cover/check-xml.html. Cover Pages: Validate/Check XML
http://www.xml.gr.jp/relax/. The official site of RELAX
http://relaxng.org/. RELAX NG home page
http://www.schematron.com/overview.html. Schematron overview
http://fr.wikipedia.org/wiki/RDF. Wikipedia Definition
http://www.la-grange.net/w3c/REC-rdf-syntax/. Detailled explanation
http://www.w3.org/TR/rdf-primer/, http://www710.univ-lyon1.fr/ champin/rdf-tutorial/rdf-tutorial.html. Tutorial of RDF
http://www.cduce.org/. Official site of CDuce
http://www.lri.fr/ miachon/usexmp.html. Some example of CDuce
http://www.w3.org/2004/OWL/. Definition of OWL
Une approche événementielle pour la modélisation et la simulation de réseaux de neurons impultionnels
Book: XML par la pratique

List of Figures

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803