GRiNS DOM-interface

date: 4-10-1999
author: Joost Geurts

The GRiNS-DOM is an extension on the GRiNS-editor which allows you to modify and extend a hypermedia-document through a standard W3C XML-DOM-interface. This document will give an introduction on how to use the DOM as well as a detailed implementation specification of how the DOM was implemented.

Introduction

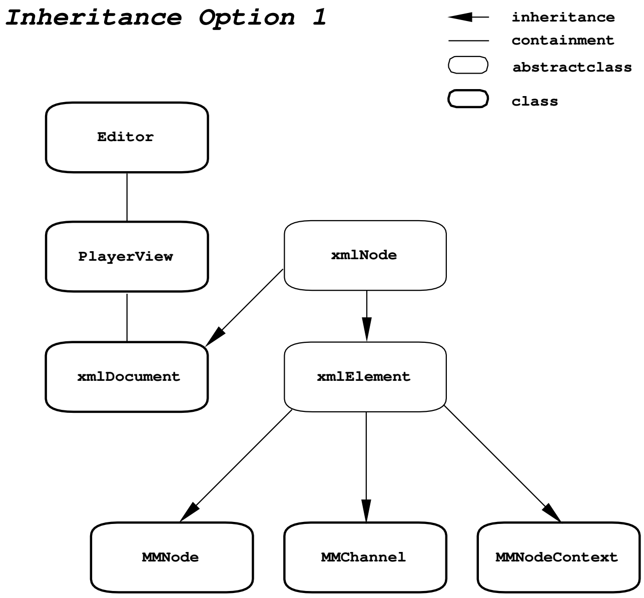
The main reason why we implemented the DOM is that we want an API on the GRiNS-editor which enables us to write independent 'intelligent' software components which are able to modify and extend the current presentation.
The internal representation of a document within GRiNS is tree-structured. The XML-DOM is tree-structured as well, therefore it seemed like a good choice to use the already existing XML-DOM instead of inventing our own, which should do about the same thing only in other terminology. Furthermore being compliant to a uniform standard is a good thing considering adjustments, extensibility and maintenance in the future.
Another important aspect of the API should be its language independency. As mentioned before we want to develop independent software components which interact with GRiNS. Being forced to use one specific language would be a bad thing since different languages have different things they are good in. CORBA provides these kinds of services and is becoming a global standard. Roughly it enables different software components running on different machines to interact with each other as if they were on the same machine.
As mentioned before the internal structure of GRiNS it tree-based. Obviously the most suitable way of altering the document is by modifying this internal structure tree. Unfortunately this structure is not SMIL-based. Instead it uses a format called CMIF which has some similarities with SMIL but is for the most quite different. So ideally the CMIF-structure should be dynamically translated to SMIL (and vice-versa). Because we already have to make some kind of translation by converting the CMIF-structure to an xml-structure we decided not to implement the SMIL-structure in the same pass. Instead, the idea is to convert the XML-structure later to its SMIL equivalent, this reduces complexity (and errors) and enables us to use the more advanced features of CMIF (if needed) which are not in SMIL (yet).

The XML-DOM

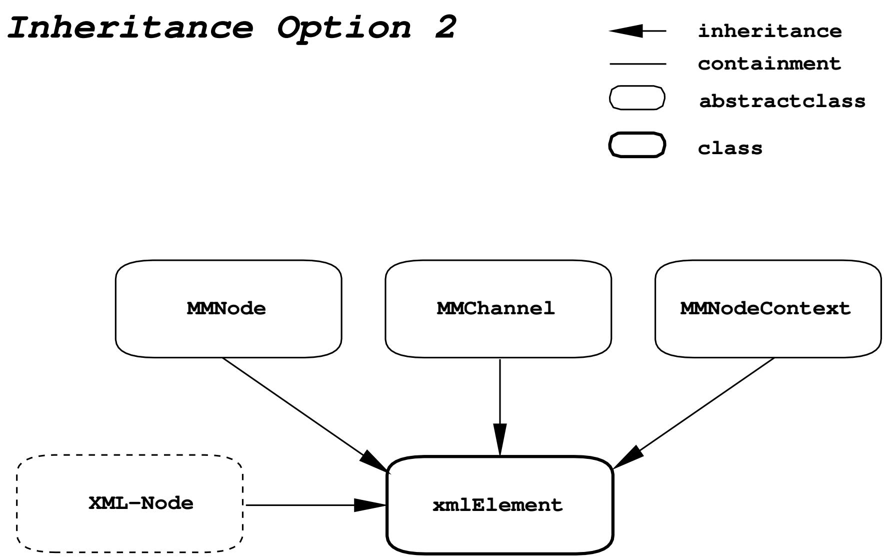
The purpose of this section is to explain the design rationale underlying the XML-DOM interface to CMIF's internal document structure. Roughly the CMIF-tree consists of nodes of type MMNode. Besides this there is a collection of MMChannels and HLinks stored in the MMContext class, which is not in the tree. An MMNode contains different kind of attributes describing the type of node and its properties and can have child-nodes. The XML-Element provides the DOM-interface to the MMNode as well as the MMChannel definitions and the HLinks. The latter two are a little more complex than the MMNode so I will leave them for later.

The thing to do on the level of translating a MMNode to an XML-Element is providing the MMNode with an alternative interface to the same data. In other words the new MMNode should behave like an MMnode, but also as an XML-Element. In OO-concepts this suggests an inheritance-relation. In this case there are some alternatives to choose from.

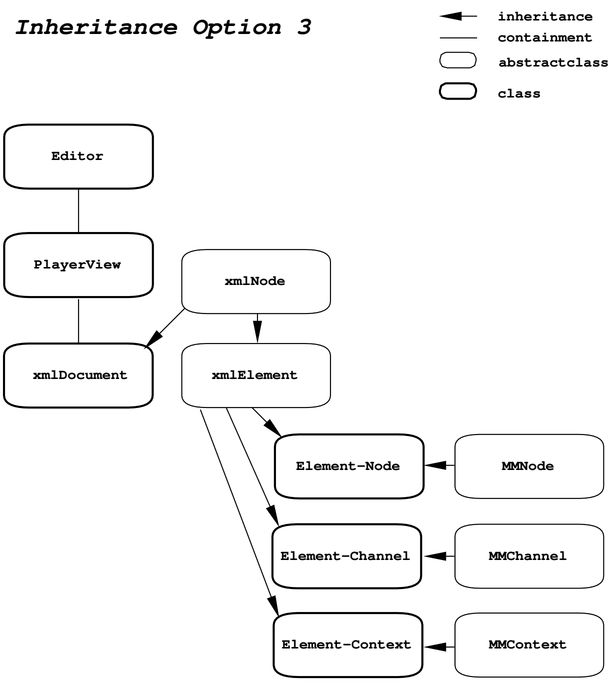
- let the old MMnode inherit a new created class XML-Element



- create a new class XML-Element which inherits MMNode



- create a new class MMNode/XML-Element which inherits both the XML-Element and the MMNode class.



The most easy option to dismiss is the second one. In this case XML-Element inherits mmnode, MMNode and mmcontext. Since they are three diffrent kind of things it is not a logical thing to do. Besides this, practically it is also not a good idea since they have common attribute names which cause problems when inheriting.

The advantage of the third one is that in this case your quite independent from the MMNodeMMNode class. Being independent is a good thing. Unfortunately this option has the bad side-effect that already created mm-node-instances in the tree are real MMNodes and do not behave like XML-Elements. When you want to treat such node as an XML-Element it wouldn't work because it doesn't inherit from the XML-Element class. To solve this problem we should everywhere in the code where an MMNode is created create a MMNode/XML-Element instance instead. To do this we have to change already existing code and by doing this lose the advantage of being independent.

We choose the first option which only has some minor disadvantages. We have to modify the existing code. However this modification are mostly independent additions and therefore maintainable. Nevertheless we tried to keep the modifications/additions to an absolute minimun, by implementing fuctionality on the XML-Element level as much as possible.

CORBA interface

Since we do not want to commit to a certain programming language to write our 'intelligent' components in we need some generic interface to access our API. Besides this we want the Client-application to be a diffrent process than the GRiNS-Server. CORBA provides these services. We have split the CORBA code into two parts. Namely the generic XML-Dom interface and

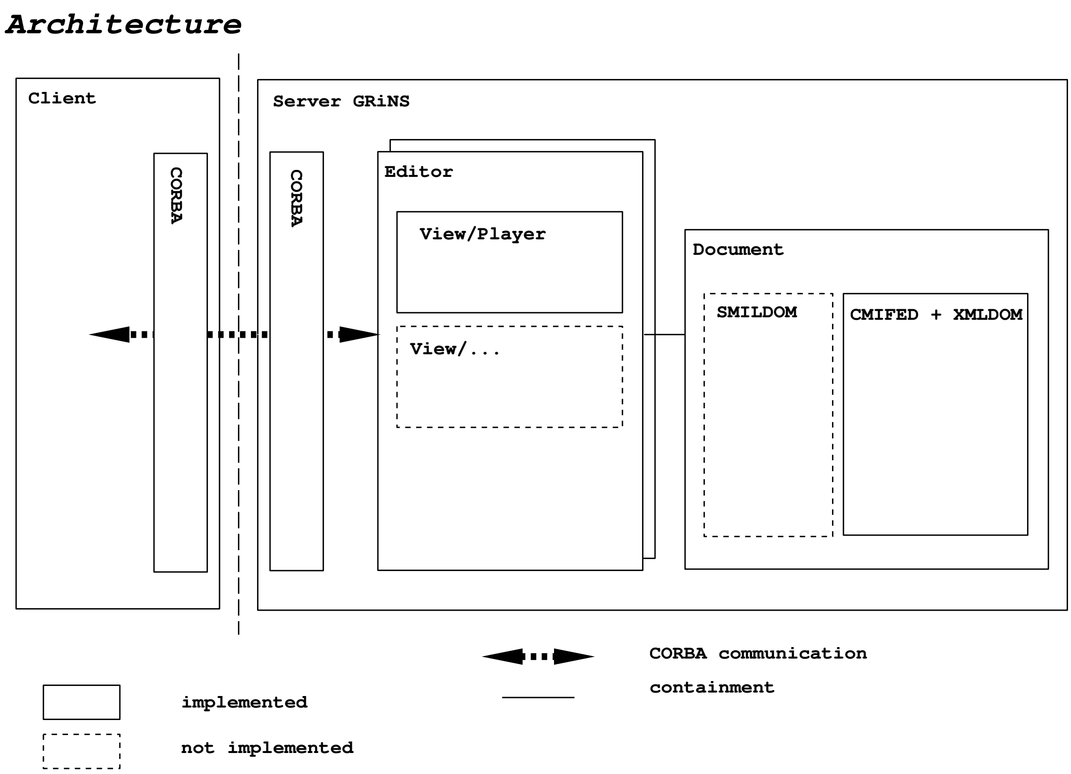
the GRiNS specific interface. The GRiNS interface has an attribute document which refers to the XML-DOM idl.
The CORBA implementation happens in three files: GRiNSAdapter.py, DomAdapter.py and GRiNSReactor.py. The first two specify the implementation for the objects used in the interfaces. These files basically contain references to the ‘real’ objects in the XML-DOM or in GRiNS which get packed into a CORBA object so it can be transmitted over the network. The third one, GRiNSReactor.py is only used to hang the CORBA server into the event loop of GRiNS.

Note that the CORBA interface is only optional. The XML-DOM can also be used directly, e.g. for python scripting.

To do

- Fully implement the interfaces.
- Error-Handling
- Dynamic CMIF-SMIL conversion

Implementation overview



Client

When using the grins-DOM in a client-program a few things are important. First of all you need the following:

- CORBA support for the programming language you use.
  (get it here: http://www.objenv.com/cetus/oo_object_request_brokers.htm)
  We used Fnorb 1.0 (http://www.dstc.edu.au/Products/Fnorb/)
- org_w3c_DOM_idl files
  (http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/idl.zip)
- GRiNS.idl
  cvs repository: /ufs/jrvosse/lib/CVSROOT/grinsDOM/CORBA/GRiNS.idl,v
- And the GRiNS editor (CORBA version)
  (Code needs to be merged with main GRiNS branch...)
  Checkout GRiNS branch with tag “corba-dom”
  cvs repository CORBA code, see /ufs/jrvosse/lib/CVSROOT/grinsDOM/

Generating stubs and skeletons

Take a look at the ‘GRiNS.idl’ file and the org_w3c_DOM_idl’s (note: we modified the w3c DOMstring.idl, replacing the wstring by a plain string. This should be easy to fix in Unicode-supporting Python 1.6). The IDL files specify the functionality of the DOM in general OO-terms and serve as an input-file to the idl-compiler. The idl-compiler generates platform-specific code, so called skeleton- and stub- files who take care of the connection to the GRiNS-server. The process of generating stubs and skeletons differs a little for all platforms, so giving a general description on how to do this is not possible. The documentation of your CORBA implementation should tell you how. For Fnorb, we used:

```
fnidl GRiNS.idl
fnidl org_w3c_dom_idl/org-w3c-dom.idl
```

Connecting

When you have generated the skeleton and stubs-files and have a running GRiNS-server, you are ready to make a connection to the GRiNS-server. Again, the way to connect to a server differs a little among different programming-languages, mainly syntax though. Currently, the server stores the object-reference of the main server object in idl-terms in a file called ‘.GRiNS.ref’ in the home directory of the person who started the GRiNS server. This mechanism should be replaced by one that is based on a proper CORBA nameserver

This is how it could be done in Python:

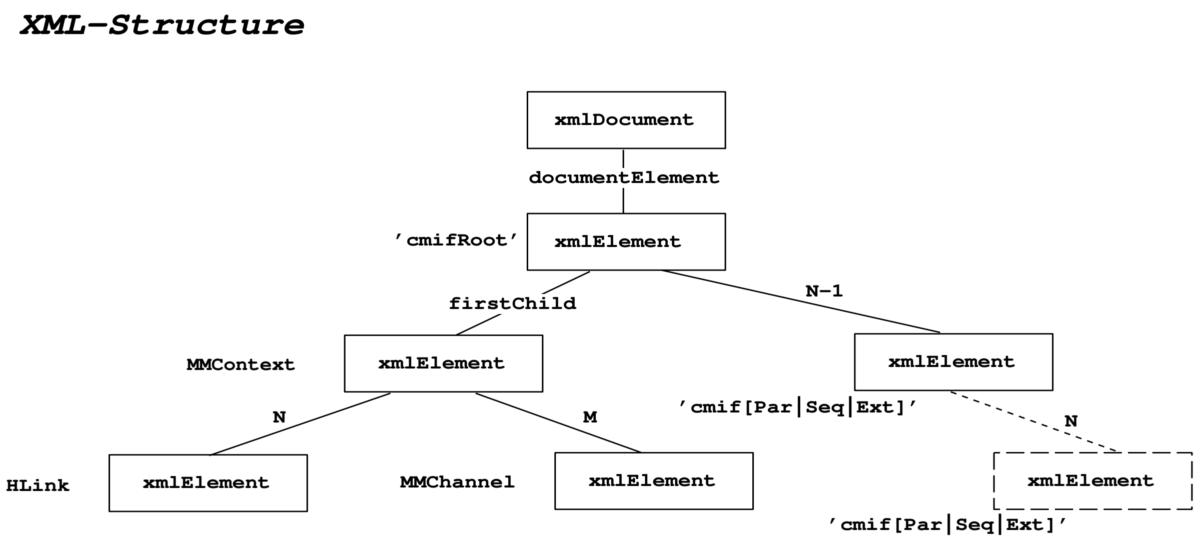
```
import grinsDOM.CORBA.GRiNS
import grinsDOM.CORBA.org_w3c_DOM
url = "/ufs/geurts/data/src/grinsDOM/test/cezanne.smil"

def main(argv):
    global url
    import sys
    import os

    file = os.path.expanduser('~/.GRiNS.ref')
    sys.stderr.write('Reading server object reference from %s\n'%file)
    f = open(file, 'r')
    stringified_ior = f.read()
    f.close()

    from Fnorb.orb import CORBA
```

```
orb = CORBA.ORB_init(argv, CORBA.ORB_ID)
grins = orb.string_to_object(stringified_ior)

id = 'IDL:oratrix.com/GRiNS/Server:1.0'
if not grins._is_a(id): raise 'This is not an %s server!'%id

editor = grins.openEditor(url);

print "Created editor for " + editor.getURL()
...
```

You can now use the (remote) editor as if it were your local copy.

Navigating through the document.

The document you navigate through is an 'XML-translation' of the CMIF-format which is used to represent the document within the GRiNS editor. We tried to make the translation as uniform as possible, but there are a few trade-offs.
MMChannel, HLinks and MMContext are not really represented in the CMIF-tree as nodes. Instead they are attributes of the MMContext which is an attribute of the root-node. In XML everything you use to represent the document should be in the document tree as well. We did this by creating an Element for the context consisting of several Elements, which are either MMChannel definitions or HLinks. The context-Element is the first child of the root-node of the document
Below you will find a schema representing the global structure of the Document in XML-DOM terms. The squares represent nodes, the type of the node is written within the square. The text in front of the squares represents their CMIF-translations.



Modifying the document

modifying the document is quite straight forward there are however a few things you should be aware of.

- Creating Channel Definition
  When you want to create a MMChannel in GRiNS you need at least two arguments namely,

a name and a type The ‘createElement’ function in the DOM however only takes one tagName (which is ‘channel’ in this case) as argument. The necessary arguments should be passed to the Element through its attributes. So before adding a Channel-element to the root-element you should have set the attributes ‘name’ and ‘type’.

- Creating a Hyperlink
  The same thing as Creating Channels apply’s for Hyperlinks. Before adding a Hyperlink-element to the root-element you should have set its attribute ‘to’ and ‘from’

Setting attributes

Every attribute in the DOM is represented as a String. There are however some attributes which take more complex formatted strings such as tuples, lists, longs, integers. For the time being we use the Python String representation. The following attributes have a special form:

- "duration" - long - “X”
- "winsize" - 2tuple - “(X,Y)”
- "base_winoff" - 4tuple - “(X1,Y1,X2,Y2)”
- “anchorlist” - list of 3tuples - “[(ANCHORID, TYPE, (X1,Y2,X2,Y2)]”
- “to”, “from” - 2tuple - “(EXTUID, ANCHORID)”

Example:

```
...
document = editor.getDocument()
root = document._get_documentElement()
context = root._get_firstChild()

newchannel1 = document.createElement('channel')
newchannel1.setAttribute('base_window', ' root-layout ')
newchannel1.setAttribute('base_winoff', '(20, 20, 600, 400)')
newchannel1.setAttribute('name', 'my_image_channel')
newchannel1.setAttribute('type', 'image')
context.appendChild(newchannel1)

nwElementSeq = document.createElement("seq")
root.appendChild(nwElementSeq)

nwElementExt1 = document.createElement("ext")
nwElementExt1.setAttribute("channel", "my_image_channel")
nwElementExt1.setAttribute("name", "my_image1")
nwElementExt1.setAttribute("file", "cezanne.bay-estaque.jpg")
nwElementExt1.setAttribute("duration", "15")
nwElementSeq.appendChild(nwElementExt1)
print editor.toxml()
return 0
document = editor.getDocument()
root = document._get_documentElement()
context = root._get_firstChild()

newchannel1 = document.createElement('channel')
newchannel1.setAttribute('base_window', ' root-layout ')
newchannel1.setAttribute('base_winoff', '(20, 20, 600, 400)')
newchannel1.setAttribute('name', 'my_image_channel')
newchannel1.setAttribute('type', 'image')
context.appendChild(newchannel1)
```

```
    nwElementSeq = document.createElement("seq")
    root.appendChild(nwElementSeq)

    nwElementExt1 = document.createElement("ext")
    nwElementExt1.setAttribute("channel", "my_image_channel")
    nwElementExt1.setAttribute("name", "my_image1")
    nwElementExt1.setAttribute("file", "cezanne.bay-estaque.jpg")
    nwElementExt1.setAttribute("duration", "15")
    nwElementSeq.appendChild(nwElementExt1)
    print editor.toxml()
    return 0
```

More examples can be found in the CVS repository ../grinsDOM/test/