Suport de XML/MPEG7 per a una llibreria de processat d'àudio i música

David García
vokimon@jet.es

30 de maig de 2002

Índex

Índex de figures

Índex de taules

Índex de llistats

Capítol 1

Abstract

La introducció de l'estàndard MPEG7, que es refereix a la codificació en XML de descriptors multimèdia, fa que sigui útil dotar a una llibreria de processat d'àudio i música de la capacitat de treballar amb XML. L'objectiu d'aquest treball és dissenyar, per a una llibreria d'aquest tipus, un sistema activació i passivació, en XML, de les seves dades de processament, i, habilitar l'ús d'XML per altres funcionalitats com ara definició de sistemes, configuració de processos i represa de càlculs posposats. S'ha volgut minimitzar la feina a fer pels usuaris de la llibreria i els creadors de nous objectes, tot donant la possibilitat de personalitzar els formats de sortida.

Capítol 2

Resum

Molts factors fan que sigui útil dotar a una llibreria de processat d'àudio i música de la capacitat de treballar amb el llenguatge de marques XML. Dos dels més importants són les seves qualitats com a interfície entre sistemes diferents, i la introducció de l'estàndard MPEG7, que es refereix a la codificació de descriptors multimédia en XML.

Alguns llenguatges de programació, en tenir mecanismes d'introspecció, poden passivar en XML els objectes del model a partir de la seva estructura. Però, la llibreria en qüestió, ha de implementar-se en C++, un llenguatge sense mecanismes propis d'introspecció, per construir sistemes en temps real que requereixen computació intensiva.

L'objectiu d'aquest treball és dissenyar per aquesta llibreria un sistema d'activació i passivació en XML dels objectes que intervenen en el processat amb molt poca intervenció de l'usuari de la llibreria.

Per això, es fa, primer, un estudi de les tecnologies existents relacionades amb XML i que es podrien fer servir amb la llibreria. Després, es descriu la llibreria de processat on s'integra el projecte. També es fa un estudi de algunes tècniques i paradigmes de programació que ajuden a suplir la mancança d'introspecció a C++.

La part pràctica del projecte és la implementació del suport XML per la llibreria amb diverses funcions: Configuració i definició de sistemes de processat, entrada i sortida de dades, control, monitorització i passivacio/activació del procés en estats intermedis.

Capítol 3

Introducció

3.1 Trets generals

CLAM (*C++ Library for Audio and Music*) és una llibreria de suport per aplicacions d'àudio musical. El tipus d'aplicacions que suporta són aplicacions que fan processament d'àudio a baix i alt nivell (nivell de senyal i nivell semàntic).

CLAM ofereix el necessari per encapsular i combinar algorismes de processat, i gestionar el flux de dades entre ells. També ofereix molts algorismes de processament ja implementats, Defineix tipus de dades de processament molt heterogenis i dóna la possibilitat de crear-ne de nous. Proporciona una capa d'abstracció del hardware d'àudio multiplataforma i integrada amb el control de flux. També ofereix una capa de visualització que, sense tacar el model de processament, dóna la possibilitat de sincronitzar amb el model vistes construïdes amb múltiples toolkits gràfics com QT, FLTK, OpenGL...

CLAM es descriu amb més deteniment al capítol 5.

Aquest projecte es presenta com a una part de CLAM. El projecte va començar al mateix temps que el disseny de CLAM. No és pas un projecte accessori sinó que amb ell s'ha contribuït en el disseny i la implementació de la mateixa llibreria. La llibreria

estava construïda des de zero i sovint també he contribuït a crear infraestructures bàsiques que encara no eren disponibles.

XML és un estàndard que permet definir un llenguatge de marques per representar dades d'aplicació. En ser un format jeràrquic pot representar dades estructurades. També és un format basat en text, la qual cosa fa possible que els humans el poguem llegir i editar amb un simple editor. Entorn a XML hi ha un seguit de tecnologies que amplien la seva funcionalitat. XML i aquestes tecnologies associades es descriuen i valoren al capítol 4.

Pel disseny i la implementació de la llibreria CLAM en general i del suport XML en concret s'han fet servir moltes tècniques de disseny i implementació, com ara programació orientada a objectes, patrons de disseny, programació genèrica, i metaprogramació amb templates. Aquests paradigmes i tècniques, s'expliquen al capítol 6 de la memòria.

Aquest projecte va ser presentat al MOSART2001, *Workshop on Current Research Directions in Computer Music* [48].

3.2 Definició dels objectius

En un sistema de processat a alt nivell generalment hi ha una part que analitza àudio i n'extreu atributs. Aquests atributs poden ser transformats i finalment resintetitzats. L'anàlisi, l'edició i la resíntesi es poden fer internament en una mateixa aplicació o bé, en aplicacions diferents. En aquest cas XML és una bona eina per fer d'interfíce entre aquestes aplicació, sobretot tenint en compte que MPEG-7 estandarditza com es codifiquen en XML aquest tipus de descriptors.

A XML se li poden assignar altres funcionalitats diferents tal i com representa la figura 3.1: definició i configuració dels sistemes, resguard i recuperació de l'estat del processament, mecanismes de reutilització...

L'objectiu del projecte és donar a l'usuari de la llibreria eines per treballar fàcilment

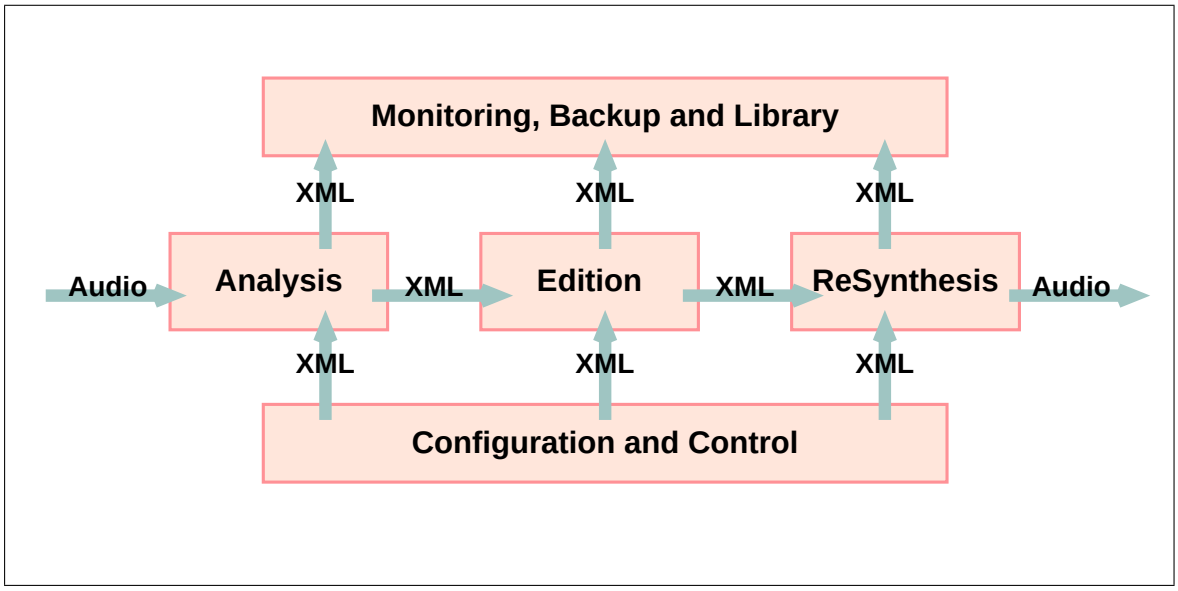

Figura 3.1: Papers que pot adoptar XML a CLAM

amb XML. Donat que la llibreria és un framework que l'usuari pot ampliar amb els seus propis objectes, no només cal implementar la serialització XML dels objectes de la llibreria, sinó, a més, cal facilitar que l'usuari pugui implementar la serialització dels seus propis objectes.

El suport XML ha de tenir les següents característiques:

- Ha de comportar la mínima intervenció per part de l'usuari de la llibreria.
- No ha de comportar càrrega al sistema de processament. L'eficiència del processament és prioritària.
- Ha de permetre personalitzar el format XML obtingut.
- Cal facilitar el reaprofitament del sistema per fer-ho servir amb altres formats, concretament SDIF[84].

Aquesta llista conté, per ordre de prioritat els objectes dels que interessa i es factible tenir una representació XML:

- Tipus bàsics i estructures de dades genèriques que serviran per construir objectes més complexes
- Dades de processament
- Configuracións
- Estat dels processos
- Definicions de xarxes de processos
- Estat del flux

Els tres primers són d'implementació obligada perquè seran els que caldrà fer servir a la data d'acabament del projecte. A l'acabament del projecte, CLAM no donarà suport a les funcionalitats per a les que serveix l'XML en els altres tres casos (definició i resguard del sistema amb fitxers XML). Aquestes funcionalitat estan planificades per finals de l'any 2002 i principis del 2003.

Altres funcionalitats dessitjables són:

- Que el suport XML sigui trasparent a l'usuari
- Suport a referències creuades dintre del document.
- Suport a objectes polimòrfics.
- Generar l'XMLSchema associat al format
- Respectar l'espai de noms

Cal dir que com CLAM es va començar a fer al mateix temps que aquest projecte, hi havia funcionalitats que no estaven disponibles. És també objectiu d'aquest projecte implementar aquestes mancances en la infraestructura del sistema i col·laborar amb els altres desenvolupadors en el disseny global i en l'establiment de metodologies.

Capítol 4

Anàlisi dels estàndards relacionats amb XML

En aquest capítol es fa un anàlisi de l'estàndard XML i diverses tecnologies relacionades amb ell. De l'anàlisi, es podrà detectar aquelles que poden fer servei en el decurs del projecte.

4.1 El format XML

4.1.1 Concepte i motivació

XML[31] és un format per representar dades d'una forma portable. XML té característiques comunes amb l'HTML, el llenguatge de marques per a documents d'hipertext que es fa servir a la WWW. No és quelcom casual. XML es va dissenyar amb la intenció que arribés al nivel de popularitat de l'HTML.

La gran potència que té el llenguatge HTML ha estat deguda a que permet compartir els mateixos documents d'hipertext entre diferents aplicacions:

- que acompleixen diferents rols en el cicle de vida del document (edició, servidors, fullejadors...),
- que han estat desenvolupades per diferents proveïdors d'aplicacions,
- i que s'executen en diferents plataformes.

Tot això potenciat per una entitat estandarditzadora, W3 Consortium[14], que defineix uns estàndards consensuats entre els proveïdors d'aplicacions. D'aquesta manera, s'eviten les iniciatives unilaterals que van fer perillar aquest estàndard en la seva primera etapa de maduresa.

HTML està limitat al domini de les aplicacions de hipertext, i no és útil per un altre tipus de document, és a dir, no es pot representar amb HTML qualsevol tipus de dades. Malgrat això, el nivell d'interoperabilitat aconseguit per l'HTML és desitjable per qualsevol altre domini d'aplicació.

És per això que va sorgir la iniciativa del llenguatge XML. XML és un llenguatge de marques, com HTML, però, que es pot adaptar a un tipus de dades qualsevol. De fet, XML per sí mateix no fixa el format dels document sinó només especifica els seus elements organitzatius. Una aplicació d'XML en concret ha de fixar com adaptar aquests elements organitzatius a les seves dades.

4.1.2 Estructura dels arxius XML

En el document d'exemple 4.1 s'observen la majoria dels elements sintàctics que serveixen per organtizar les dades: La primera línia és una capçalera que indica quina versió de XML es fa servir i quina codificació (conjunt de caracters). Tenim un exemple de comentari (línia 2), i a partir d'aquí una organització jeràrquica de tags: el que anomenem elements XML.

Els elements són els que estableixen l'estructura jeràrquica. `descriptorset` és l'element arrel. El seu contingut acaba amb el tag `</descriptorset>`. `target` és un atribut d'aquest element que pren com a valor `vector`. L'element arrel també conté

```
<?xml version='1.1' encoding='ISO-8859-1' ?>
<!-- A comment -->
<descriptorset target="vector">
    <descriptor name="average">23.1</descriptor>
    <descriptor name="median">30.2</descriptor>
    <descriptor name="deviation">23.1</descriptor>
    <vector cardinality="9">
        23 235 -35 -23 -46 34 25 -63 24
    </vector>
    <process name="centrifugation" />
</descriptorset>
```

Llistat 4.1: Exemple simple de document XML

un seguit de elements dins seu. Cadascun té algun altre atribut i contingut pla, que és el que queda fora dels angles que emmarquen els tags.

L'element `process` (linia 10) és un element buit que només conté attributs, i, ni conté altres elements, ni text pla. Aquests elements buits no necessiten un tag d'acabament, només posar una barra al final del tag d'inici.

4.1.3 Característiques

Al mateix document d'exemple (llistat 4.1) es pot observar:

- que és un format basat en text molt semblant a l'HTML, i
- que les dades estan organitzades d'una forma jeràrquica.

Podem adaptar a les dades del domini tant els noms dels elements i dels atributs com la pròpia estructura jeràrquica. Això ens dóna molta flexibilitat, però per poder manegar-la, cal eliminar algunes ambigüitats que hi havia en l'HTML. En HTML, aquestes ambigüitats es resolien gracies a que els parsers coneixien el domini. XML ha de ser independent del domini i no es poden fer aquestes suposicions, així que s'imposen algunes restriccions:

- Només hi ha un element com a arrel.
- Tot element ha de tenir un parell d'inici i final o ser buit.
- Els tags d'inici i final han d'estar ben balancejats.
- Els noms són sensibles a majúscules.
- Els valors dels atributs han d'estar entre cometes (dobles o simples).

4.1.4 Valoració

XML és un format llegible i modificable per humans i, al mateix temps, llegible i modificable per aplicacions. On té més sentit fer servir un format llegible pels humans, és en aquelles situacions on un usuari humà ha de comunicar a un programa una informació estructurada en forma de fitxer. L'exemple més clar són els fitxers de configuració.

En ser un format basat en text és inherentment menys eficient, en volum i en temps de processat doncs cal interpretar el text.

Així doncs, en altres casos en que no es requereix llegibilitat, la seva utilitat és molt discutible. Però, encara té avantatges com a format: la seva estandardització, l'expressivitat estructural i la capacitat de fer interactuar sistemes.

Redundància dels elements estructuradors

Sense deixar de ser un format llegible, el format en sí podria haver estat quelcom més eficient, sinó fos per la restricció de disseny de mantenir la compatibilitat amb SGML.

XML és un format de text amb molta redundància. Els tags ocupen molt per ells mateixos. No només és la longitud del nom del tag, sinó que, a més, normalment es repeteix al obrir i al tancar. Els dissenyadors de l'estàndard ho justifiquen dient que

així és molt més llegible, però igual de llegible pot ser, per exemple, un format basat en claus *alla* C com els que s'han vingut fent servir fins ara per codificar informació estructurada (vegeu el llistat 4.2).

Un llenguatge estructurat amb claus pot proporcionar la mateixa meta informació que XML i és més compacte. Ambdós formats es poden llegir amb la mateixa facilitat si estan ben indentats, i els dos es llegeixen amb dificultat si no estan indentats correctament.

Més endavant veurem algunes abstraccions del format XML que fan possible tenir codificacions d'aquest estil mantenint el model conceptual.

```
?xml {@version='1.1' @encoding='ISO-8859-1'}
# A comment
descriptorset {
    @target="vector"
    descriptor {@name="average" 23.1}
    descriptor {@name="median" 30.2}
    descriptor {@name="deviation" 23.1}
    vector {
        @cardinality="9"
        23 235 -35 -23 -46 34 25 -63 24
    }
    process {@name="centrifugation"}
}
```

Llistat 4.2: Format alternatiu amb menys redundància

Dades numèriques i multimèdia

Molts articulistes que parlen sobre XML, el venen com una solució global als problemes de representació de la informació. En aquest sentit cal ser molt escèptic, no és del tot cert. Tot i que XML es pot adaptar a qualsevol tipus de dada, no sempre és el més adient per contenir alguns tipus de documents.

Els continguts típicament binaris, com ara imatges o so, són molt ineficients de codificar en XML. La representació decimal, que es fa servir a les primeres propostes

MPEG7 per a les dades com els espectres és molt dolenta: ocupa més i la codificació i descodificació dels nombres es porta un temps d'execució desorbitat.

Una altra alternativa és la codificació amb eines del tipus `uuencode` que codifiquen un flux binari amb caràcters imprimibles ASCII. Aquesta aproximació és molt més eficient en espai, i bastant més eficient en temps de processament.

El fet és que, amb aquesta codificació, estem perdent la llegibilitat. Un flux binari codificat en ASCII llegible no ho és gens de llegible per humans. Generalment, si estem plantejant-nos una codificació d'aquest tipus, és perquè no ens interesa gens la llegibilitat. Llavors, l'única raó per incloure les dades binàries amb la resta del document XML és per mantenir l'estructura dintre del document.

En aquest cas, l'alternativa és excloure la informació binària i substituir-la per referències a recursos externs. Els recursos binaris externs i el XML amb l'estructura es poden juntar en un arxiu comprimit.

En resum, cal anar amb cura quan representem amb XML informació de natura binària, donat que pot resultar en una pèrdua d'eficiència en espai i temps quan es treballa amb grans volums de dades. També cal considerar que malgrat no sigui pràctic per aplicacions que necessiten treballar eficientment amb grans volums de dades binàries, representar aquesta informació binària en format llegible resulta molt útil de cara als desenvolupadors, que de fet són els nostres usuaris.

Implementacions deficients

El processament del format XML a les implementacions existents és lent i costós en memòria. Això és degut a l'API estàndarditzadas, les diferents capes del DOM (Document Object Model). El DOM indueix a fer una implementació que representi en memòria l'estructura del document.

Els implementadors i els estandarditzadors s'han adonat que aquesta no és la millor forma de fer anar XML i estan prenen accions correctives al respecte que es comenten amb més detall a l'apartat 4.8.

4.2 Resolució de colisions de noms: Namespaces

Cada format XML d'usuari fa servir els seus propis noms pels elements, atributs... Quan dos formats concrets han de conviure és possible que es produeixi col·lisió en els noms. Per aquesta raó, es fan servir espais de noms (namespaces).

```
<?xml version="1.0"?>
<!-- initially, the default namespace is "books" -->
<book xmlns='urn:loc.gov:books'
      xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <title>Cheaper by the Dozen</title>
    <isbn:number>1568491379</isbn:number>
    <notes>
      <!-- make HTML the default namespace for some commentary -->
      <p xmlns='urn:w3-org-ns:HTML'>
          This is a <i>funny</i> book!
      </p>
    </notes>
</book>
```

Llistat 4.3: Exemple d'ús dels Namespaces

Un espai de noms s'identifica per una URI. Una URI és un identificador únic en la xarxa pot ser una URL que apunti a un recurs existent, o una URN que simplement es fa servir per donar noms únics. Les URN, tot i que no han de ser URL's, sovint es fa servir sintaxis semblant a les URL's per assegurar la seva unicitat ja que, en dur una DNS, s'assegura que l'entitat a qui pertany el domini DNS controla aquesta unicitat.

Els espais de noms s'especifiquen per a un element i els seus fills. Cal afegir a aquest element alguns atributs especials. Els fills hereten les definicions de espais de noms, però, poden redefinir-los.

Per definir el namespace per defecte cal posar l'atribut `xmlns` amb la URI del namespace com a valor. Tots els noms d'aquest element (inclòs el del propi element) i els seus fills es cerquen a aquest espai de noms si no es diu el contrari.

Per que dos espais de noms convisquin cal definir espais de noms alternatius. Això es

fa, afegint-hi un atribut `xmlns:foo` amb la URI d'aquest espai de noms alternatiu. `foo` serà el prefixe que haurem de posar als noms que en pertanyen.

4.3 Definició de formats d'aplicació: XML-Schema

4.3.1 Llenguatges de definició

XML és només un marc per definir formats concrets. L'estàndard només especifica els elements constructius. Així doncs, cal que cada aplicació d'XML defineixi el seu format concret que implica especificar:

- quins noms es fan servir pels elements i atributs
- quina estructura té el seu contingut

Aquesta és la funció que fan els llenguatges de definició. El primer llenguatge de definició que va estandarditzar el W3C va ser el DTD. Els DTD's sovint es van quedar curts per la seva poca expressivitat i la complexitat de la seva sintaxi (SGML) si es volia fer especificacions complertes.

Actualment els DTD's han quedat desfasats i s'ha establert un altre llenguatge de definició: XML-Schema.

4.3.2 Conceptes generals

L'estàndard XML-Schema[45] especifica una sintaxi pels documents que defineixen el format XML d'aplicació (XML-Schema Definitions o xsd's) i la forma com es relacionen amb aquesta definició els documents XML que l'instancien (XML-Schema Instances o xsi's).

XML-Schema es basa en una idea molt semblant a la definició de variables i tipus en llenguatges de programació tipificats com ara C. Es distingeix entre tipus d'objectes

simples i tipus d'objectes compostos. Els objectes simples són els que no poden contenir dins cap atribut ni element, es representen amb text pla. Els objectes compostos són els que, a més, en poden contenir atributs i elements.

Un esquema XML es composa doncs de definicions de tipus i definicions d'atributs i elements que pertanyen a aquests tipus. L'esquema del llistat 4.4 defineix un tipus de document sense fer servir declaracions de tipus.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org"
            xmlns="http://www.books.org"
            elementFormDefault="qualified">
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:element name="Book">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element ref="Title" type="xsd:string
                          minOccurs="1" maxOccurs="1"/>
                        <xsd:element ref="Author" type="xsd:string
                            minOccurs="1" maxOccurs="1"/>
                    </xsd:sequence>
                 </xsd:complexType>
             </xsd:element>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Llistat 4.4: Schema amb tipus anònims

La mateixa descripció que en podem fer per a un objecte (element o atribut) la podem associar amb un nom tipus. Aquest nom de tipus es pot aprofitar per definir aquesta mateixa estructura d'objecte arreu (llistat 4.5).

Si no s'especifica el contrari, cada membre d'un objecte o tipus pot apareixer o no multiples vegades en un ordre indistint. Però, es pot forçar una seqüència, opcionalitat, cardinalitat, alternatives...

L'estàndard també preveu com utilitzar tipus ja definits per definir-ne de nous, fent servir mecanismes d'herència. L'herència no només permet estendre un tipus sinó

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org"
            xmlns="http://www.books.org"
            elementFormDefault="qualified">

   <xsd:complexType name="BookType">
       <xsd:sequence>
           <xsd:element name="Title" type="xsd:string
               minOccurs="1" maxOccurs="1"/>
           <xsd:element name="Author" type="xsd:string
               minOccurs="1" maxOccurs="1"/>
       </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:element name="Book" type="BookType" />
        </xsd:complexType>
    </xsd:element>

</xsd:schema>
```

Llistat 4.5: Schema amb noms de tipus

també restringir o redefinir alguns membres. A més, també es pot fer servir l'herència a tipus simples per crear altres tipus simples.

L'exemple 4.6 defineix una extensió del tipus `BookType` amb el nom `MultiAuthorBookType` per que tingui.

L'herència també serveix com a mecanisme per agrupar un seguit de tipus com sota una sola denominació de tipus, que es pot ampliar en altres documents de definició.

En el cas anterior si especifiquem que el tipus `BookType` és `abstract`, podriem fer servir indistintament qualsevol dels tipus de llibre.

Els noms del tipus, dels tags i dels atributs que es defineixen a un esquema van a parar a un espai de noms que fa de receptacle de les definicions que en fem. És el namespace definit a l'atribut `targetNamespace` de l'element arrel de l'esquema. Hom quan defineix un esquema pot importar un altre esquema existent i ampliar-ho.

```
<xsd:complexType name="PricedBookType">
    <xsd:complexContent>
        <xsd:extension base="BookType">
           <xsd:sequence>
               <xsd:element name="" type="xsd:string
                   minOccurs="1" maxOccurs="unbounded"/>
           </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="BookStore">
    <xsd:complexType>
        <xsd:element name="Book" type="PricedBookType" />
    </xsd:complexType>
</xsd:element>
```

Llistat 4.6: Extensió d'un tipus

XML-Schema també defineix com referenciar l'esquema des de el document instància. Aquestes referències són per indicar quin esquema fa servir la instància o explicitar el tipus d'un node abstracte.

Especificar el tipus als nodes abstractes és important per poder saber com llegir el tipus.

4.3.3 Valoració de la tecnologia

XML-Schema és una eina potent perquè permet definir amb múltiples tècniques el model de dades que representa el format XML d'aplicació. Fa ús del concepte de tipus i tècniques utilitzades a l'orientació a objecte que permet estalviar feina de definició, reutilitzant tipus ja definits per definir-ne nous o deixant portes obertes a l'ampliació futura i fins i tot per part de tercers.

Permet construir models de dades estàtics. Això vol dir que, tot i que definim els objectes d'una manera orientada a objecte, no estem definint cap tipus d'operació per modificar-los.

Pel mateix motiu cal dir que la modelització que es fa és una modelitzacio sintàctica de les dades. Això no vol dir que sigui dolent, de fet el model sintàctic és el que s'ha cercat amb les tecnologies XML. Models semàntics ja n'hi ha hagut, el problema ha estat que fins ara comunicar dos sistemes amb un mateix model semàntic no havia estat una qüestió clara. XML ha començat a partir de la sintaxi i, en part, ha estat la raó del seu èxit.

Cal tenir en compte que, per fer una definició acurada de la sintaxi que volem, cal arribar a un nivell de detall molt prim que normalment implica un document de definició bastant gran.

A més, XML-Schema és a la vegada una instància XML del seu propi esquema. Per això, hereta els vicis de XML:

- Fa servir un **format de text** que no té ni de lluny la compactació d'un format binari.
- Els documents XML tenen molta **redundància** encara i sent un format de text.
- El processament XML a les implementacions actuals és **lent i costós en memòria**.

4.4 Model conceptual: InfoSet

4.4.1 Concepte i necessitat

Les tecnologies XML estan molt basades en la sintaxi XML. Aquesta sintaxi implica uns costos molt alts d'emmagatzemament i processat. L'XML Information Set (abreviat InfoSet)[37] és un primer pas per aconseguir independitzar les tecnologies XML de la sintaxi de tags.

L'*Information Set*, de fet, no especifica res que sigui directament usable, només descriu quina és la informació que és significativa dintre d'un document XML ben format.

Abstreu les relacions entre els elements, els atributs, el contingut, els espais de noms... Donat que aquesta abstracció és independent de la sintaxi, l'InfoSet ens permet tractar amb el model formal del document.

Aquest estàndard va aparèixer per la necessitat d'unificar nomenclatura i criteris entre els diferents grups de treball al W3 Consortium que es basen en XML. L'especificació d'XML deia com ha de ser un document XML però, no pas com aquest document és vist per l'aplicació.

Com a efecte lateral van aparèixer noves possibilitats en d'aïllar l'aplicació de la sintaxi:

- Permet fer servir una sintaxi diferent a XML per serialitzar dades XML. Qualsevol format que pugui representar aquesta semàntica es podria fer servir com dades XML de forma transparent per les aplicacions. Això permetria crear serialitzacions molt més lleugeres, ràpides i flexibles que l'actual XML.
- Permet proveir API's diferents de DOM o SAX (que s'expliquen a l'apartat 4.8) per manegar aquesta informació. No crec que sigui una bona idea crear API's indiscriminadament; una API estandarditzada hauria de ser prou, però, una reformulació de l'API[42] basada en InfoSet, podria ser interessant donat l'irregular origen de DOM, que va nèixer font d'una solució de compromís entre dos propostes propietàries amb poc criteri generalitzador,

4.5 Selecció de nodes: XPath

4.5.1 Funcionalitat

Una de les tecnologies XML més potents és XPath[29]. En XPath es basen algunes tecnologies XML, com ara XSLT, XPointer i XQuery. La seva funció és referenciar un conjunt de nodes XML d'un document. La forma de fer-ho s'inspira en la forma en que s'especifiquen els encaminaments a recursos en els sistemes de fitxers, però, amb

refinaments entre els que s'inclouen comodins, expressions de filtre, eixos múltiples...

El resultat són expressions semblants a un encaminament de fitxers però molt més expressives que es fan servir tant per selecció, com per encaminament, com per comprovar la correspondència de patrons.

4.5.2 Breu descripció del llenguatge

Tota expressió XPath té sentit avaluada dintre d'un document i prenent com a base un punt d'aquest document, el node de context. És anàleg a la idea de que tot encaminament a fitxers té sentit en un sistema de fitxers i posicionats en un directori en concret: el directori de treball.

L'expressió XPath és una seqüència separada per barres '`/`' de passos de navegació per l'estructura del document. Cada pas mou la selecció de nodes a partir dels nodes seleccionats en el pas anterior. Depenent de si l'expressió comença amb barra o no, la navegació comença a l'arrel del document o al node de context, de forma similar als encaminaments absoluts i relatius dels sistemes de fitxers.

Cada pas s'especifica de la següent forma:

```
eix::test[predicat1][predicat2]...
```

Amb `eix::` s'indica una dimensió en la que moure's. Dimensió no és res més que el conjunt de nodes que volem abastar en el pas. Els possibles eixos es poden veure a la taula 4.1.

Hi ha dos eixos es poden abreviar: Si no s'especifica l'eix, es considera `child::`, i si en comptes de l'eix posem un símbol '`@`', es considera `attribute::`.

El `test` d'un pas és una primera selecció dintre dels nodes disponibles a l'eix. Pot ser un tipus de node seguit de parèntesis (`node()`, `text()`, `processing-instruction()`, `comment()`), un nom d'element o atribut (segons l'eix) o un asterisc que selecciona tots els nodes disponibles.

Eix	Nodes disponibles
`self::`	el node de context
`attribute::`	el conjunt d'atributs del node context
`namespace::`	el node namespace del node context
`parent::`	el pare del node de context
`child::`	els fills directes del node context
`ancestrors::`	el pare del node context, i els pares del pare fins a l'arrel
`descendant::`	els fills i els fills dels fills fins a les fulles del node context
`ancestrors-or-self::`	Ancestrors + self
`descendant-or-self::`	Descendant + self
`preceding::`	tots els nodes posicionats abans que el node context
`following::`	tots els nodes posicionats després que el node context
`preceding-sibblings::`	tots els nodes germans posicionats abans que el node context
`following-sibblings::`	tots els nodes germans posicionats després que el node context

Taula 4.1: Possibles eixos a un pas XPath

Es poden abreviar també algunes combinacions d'eix i test. S'indiquen a la taula 4.2. Aquestes abreviacions aproximen, un altre cop, les expressions XPath a la forma de les expressions de path de Unix.

.	self::node()
..	parent::node()
∅	descendant-or-self::node()

Taula 4.2: Abreviacions d'XPath

Els predicats són expressions que fan de filtre addicional. Cada expressió s'avalua per a cada node seleccionat, i, si l'expressió dona fals, aquest node deixa de formar part del conjunt seleccionat.

Aquestes expressions poden arribar a ser molt complexes i, fins i tot, incloure altres expressions XPath.

4.6 Transformacions: XSLT

XPath, no fa res més que seleccionar nodes. Però, seleccionar nodes és el primer pas per fer-ne transformacions. XSLT[36] fa servir XPath per fer les transformacións necessàries per passar d'un format XML a un altre.

L'intèrpret de XSLT navega pel document origen i intenta aplicar les regles de producció que es defineixen al document XSLT per anar generant un document de destí.

Les regles de producció del document XSLT estan guardades per expressions XPath. Donat un context de navegació a l'origen, la producció s'aplica al conjunt de nodes que satisfacin l'expressió XPath que la guarda.

Les produccions, en executar-se, mouen el context al node seleccionat per l'expressió XPath i generen nou contingut a la sortida tot aplicant noves regles.

Per la generació de contingut, les produccións poden obtindre el contingut del document d'origen, seleccionant-lo amb expressions XPath.

4.7 Relacions entre documents

Hi ha tot un seguit d'especificacions del W3C que es dediquen a establir les relacions entre els documents. Les seves funcions a grans trets són les següents:

XInclude: Permet incloure un document dintre d'altre a nivell de parser.

XLink: Explicita les relacions a nivell d'aplicació entre documents, porcions de documents o recursos no XML.

XPointer: Amplia XPath per poder especificar no només nodes XML sinó també un punt o interval qualsevol dintre del document.

XBase: Permet especificar una base diferent per les referències URI's relatives.

XFragment: Contextualitza un fragment de document XML per poder obtindre el mateix *InfoSet* que si estigués acompanyat per la resta del document.

A continuació es comenten XInclude i XLink que són els que són més utilitzables en CLAM.

4.7.1 XInclude

XInclude [20] (XML inclusions) té com objectiu oferir una forma estàndard de incloure un document XML dins d'un altre a nivell sintàctic. A nivell sintàctic vol dir que les aplicacions no són conscients d'aquesta inclusió ja que s'ha resolt a nivell de parsing. En altres paraules, l'*information set* del document inclòs s'afegeix a l'*infoset* del document que l'inclou i queda disponible a l'aplicació.

Aquesta funcionalitat és útil donat que permet incloure parts comunes a diferents documents reduint la redundància entre els documents i amb això el medi ocupat.

- Fent `<include href="myIncludedPage.xml" />` s'inclou en el lloc de l'element el document `myIncludedPage.xml` com a part del document.
- Afegint l'atribut `parse="text"` podem incloure text planer tot escapant els signes amb significat especial en XML.
- Es pot fer una inclusió parcial tot especificant un interval amb XPointer.
- Si introduim l'element `failback` podem especificar dins un contingut alternatiu en cas de que el recurs no es trobi disponible.
- Els atributs i elements definit en l'especificació XInclude estan definits al namespace `http://www.w3.org/2001/XInclude`.

4.7.2 XLink

XLink [19] (XML Linking Language) estandarditza la forma de establir relacions entre diferents documents XML o entre documents XML i altres recursos o entre diversos nodes d'un mateix document.

Amb XLink també es poden fer inclusions, però, a diferència de XInclude aquestes inclusions són responsabilitat de l'aplicació a l'estil del `<object>` de HTML. En tot cas, XLink permet fer relacions més subtils que no pas l'inclusió.

Sintetitza les funcionalitats que a HTML es donaven amb els tags `<a>` (per origen i desti de referències creuades), `<object>`, `<image>` i `<applet>` per inclusió d'objectes, i `<link>` per metarelacions. No es queda només en aquestes relacions sinó que, a més, generalitza qualsevol tipus de relació que es vulgui expressar en una aplicació concreta de XML.

Qualsevol element XML pot esdevenir un enllaç si li afegim els atributs que defineix XLink. Aquest atributs defineixen les característiques de l'enllaç, per exemple:

- La posició del recurs relacionat
- El moment de la travesia: En temps de càrrega, a petició, mai...
- El lloc que adoptara el recurs: Reemplaçant el recurs referenciador, incrustat en ell, en un nou contenidor, enlloc...
- El significat de cada costat de la relació
- El significat la relació
- Les parts del document afectades per l'acció de relació

Les relacions XLink ens poden ser útils per definir recursos externs, per exemple per referenciar un recurs binari que provingui de dades que hem decidit codificar de forma externa al XML, tal i com vaig comentar a la secció 4.1.4.

4.8 API's les seves implementacions

El W3C Consortium ha definit una API neutral al llenguatge de programació per treballar amb documents XML a memòria. Aquesta API es el DOM (Document Object Model) [56][55][54].

La funcionalitat bàsica del DOM és proporcionar una interfície de programació, per construir i/o editar el model d'un document XML/HTML.

DOM en comptes de funcionar amb versions, funciona amb capes (Layers). La primera capa era molt primitiva però donava les funcionalitats bàsiques per accedir i modificar el document. Existeixen també una capa 2 i una capa 3 que encara no estan completament especificades però són ampliament usades. Les capes 2 i 3 tenen també diferents moduls que els implementadors poden soportar de forma opcional i que cobreixen arees com ara manegament d'events, lectura i escritura, estils, vistes... Molts d'aquests móduls són encara esborranys i existeixen només algunes implementacions parcials.

Hem volgut fer servir la API DOM perquè és un estàndard estès i acceptat i malgrat tingui alguns problemes. El primer és que, en ser neutral al llenguatge, no especifica res de les característiques pròpies del llenguatge, i, en conseqüència, diferents implementacións en un mateix llenguatge poden ser diferents fins i tot en la interfície. A més s'ofereix una interfície redundant per donar alternatives als llenguatges procedurals i als llenguatges orientats a objectes.

Un segon problema és que el DOM bé de la convergència de diverses API's propietàries, no pas d'un procés d'estandardització sense cap compromís.

Per últim el procés d'estandardització és molt lent, DOM Level2 encara no especificava una interfície per carregar, guardar i validar documents. Es donava llibertat als implementadors amb la qual cosa tot el codi que fa servir aquesta part de la llibreria no té cap garantia de seguir funcionant.

Actualment, s'està estandarditzant DOM Level 3[54]. Aquesta especificació, finalment abastara obrir i desar documents i donarà una interfície per validar i parsejar

segons un Schema [35].

Moltes llibreries han implementat els seus parsers amb l'objectiu d'obtindre un arbre DOM a memòria amb el que es pogués treballar. Aquest procediment es costós en temps i en memòria. És per això, que va sorgir la iniciativa de crear una API per parsejar els documents mitjançant la gestió d'events: SAX[64].

Un parser orientat a events crida certes funcions d'usuari cada cop que analitza un nou element sintàctic. Aquesta aproximació, permet que les funcions d'usuari siguin les que construeixin l'arbre DOM o directament agafin la informació que l'usuari necessita sense arribar a construir el model del document a memòria.

SAX és més complex de fer servir però permet optimitzar la càrrega dels documents. El nivell 3 d'especificació del DOM, ha inclòs també el parsing orientat a events amb SAX com a model.

4.8.1 Implementacions

Donat els objectius de la llibreria CLAM, els següents factors intervindran en la tria:

- **Suport multiplataforma.** CLAM ha de suportar Windows32, Linux i MacOS-X. Altres plataformes y l'ús de codi portable també es valorarà.
- **Condicions de redistribució.** CLAM es farà servir a dos àmbits, el del software comercial y el de software lliure.
- **Usabilitat des de C++.** El llenguatge d'implementació de CLAM és C++. Facilitaria molt la feina una implementació orientada a objectes, sobretot si és en aquest llenguatge.

Cal dir també, que com que han de seguir uns estàndards recents o en evolució durant el transcurs del projecte, les valoracions que es poden fer de les diferents implementacions han anat canviant a mida que aquestes implementacions anaven millorant en qualitat.

Apache-IBM: XercesC, XalanC i XML4C

XercesC[15] és una implementació en C++ de DOM1, DOM2, SAX1 i SAX2. És una llibreria que va començar a desenvolupar IBM amb el nom de XML4C. Posteriorment, va alliberar parcialment el seu codi font. A partir de llavors el seu desenvolupament és open source i el duu a terme oficialment el col·lectiu d'Apache.

Cal dir que aquesta llibreria és la que s'ha fet servir en aquestes primeres versions de CLAM. A banda de ser C++ i open source, és molt portable. Funciona a les plataformes que CLAM necessita i moltes altres.

XerxecC segueix l'estàndard DOM molt de prop i implementa molt ràpid les altres especificacions de W3C, servint a vegades d'implementació de referència per contrastar si alguna proposta d'estàndard va ben encaminada.

XercesC té una aproximació orientada a objectes la qual cosa la fa bona per a una integració amb CLAM. El major inconvenient de XercesC és que prové d'un port d'una llibreria en Java: XML4J que va esdevenir XercesJ. Com a conseqüència d'aquest fet, trobem que gran part de la llibreria s'encarrega de simular l'entorn Java de recol·lecció de brossa i de gestió de referències. Això la fa molt més lenta i amb més empremta a memòria.

Tot i que es suposa que aquest mecanismes que simulen Java han de fer-la més consistent en el tema de memòria, la veritat és que XercesC sempre ha estat una llibreria amb molts leaks de memòria.

El grup de desenvolupadors de XercesC, ha estat desenvolupant el darrer any una interfície DOM més adaptada a C++. De fet, és molt probable que sigui la interfície oficial en la versió 1.8. Aquesta nova interfície és molt més eficient tot i que delega molta feina de gestió de memòria a l'usuari.

Un altre inconvenient derivat de ser un port des de Java és que simula alguns serveis que proporciona la llibreria estàndard de Java, en comptes d'adaptar XercesC per que faci servir els serveis anàlegs que es defineixen a la llibreria estàndard de C++.

Sovint això suposa que per integrar la llibreria amb codi C++ existent, calgui fer algunes peripècies que d'altra manera haguessin estat directes de resoldre. Això passa, per exemple amb les cadenes de text, que poguessin haver fet servir quelcom més usable amb les `string` i `wstring` de C++.

El cas que més ens ha fet patir, i que ens va fer tornar a considerar alternatives, va ser el de les fonts i destins dels fluxos de dades. En fer servir l'aproximació de Java per abstreure els destins i els orígens dels fluxos de dades, no podíem aprofitar els mecanismes propis de C++ per generalitzar-los, els streams. Evidentment XML no és l'únic format que volem suportar, i no està gens bé que la llibreria ens lligui els fluxos de dades a uns pocs, que tot i ser els més comuns (fitxer amb nom, bloc de memòria i URL) no deixen de limitar.

La implementació que fa XercesC de les característiques no estandarditzades està esdevenint una referència de cara a incorporar-les als nous estàndards DOM. Seria una llàstima que incorporessin característiques tan lligades a la implementació a una API que es caracteritza per ser neutral al llenguatge.

XercesC té una llibreria acompanyant, XalanC que implementa XSLT i XPath. Darrerament, ha implementat el seu propi parser i la seva pròpia representació DOM optimitzats per fer servir XSLT.

CenterPoint

Mogut per la necessitat de trobar alguna llibreria que estigués més integrada dintre de C++, no fa gaire vaig trobar una altra llibreria en C++ que sent fidel als estàndards DOM, implementa la API fent servir massivament la llibreria estàndard. Està desenvolupada per una consultora anomenada CenterPoint[6] tot i que és codi obert i la seva llicència és el suficientment afí a les necessitats de CLAM.

Sembla molt més neta en el disseny i ben documentada. En fer ús intensiu de la llibreria estàndard, és molt més petita.

Com a contrapunt, sembla que el seu manteniment és molt més lent i va molt enred-

erida en quant a la implementacions dels nous estàndards com ara XML-Schema.

Gnome

La llibreria libxml[16] és una llibreria molt portable per manegar documents XML. Tot i que es va desenvolupar dintre del projecte Gnome, no té cap dependència amb aquest entorn d'escriptori. Després de fer algunes proves, la vaig descartar per diverses raons:

- era una implementació en C,
- no implementava DOM sinó la seva pròpia API, i,
- la implementació estava bastant incomplerta.

Tot i així, cal reconèixer que, després d'haver pres la decisió, amb el temps ha acabant esdevenint una llibreria molt brillant i cal remarcar que hi ha un projecte, Gdome[8], que implementa una capa DOM per sobre d'aquesta llibreria amb la qual cosa ja tindríem la compatibilitat DOM.

Malauradament aquests canvis van succeir un cop havia estat presa la decissió de fer servir XercesC.

La llibreria libxml també té una llibreria germana anomenada libxslt que implementa transformacions XSLT.

4.9 Usos de XML

El format XML avui dia ha estat adoptat com a format propi de moltes aplicacions. La W3C ha estandarditzat alguns formats concrets com ara:

- XHTML: Una reformulació d'HTML 4.1 en XML

- SVG: Un format per representar gràfics vectorials escalables.
- XMath: Un llenguatge de marques per representar formules.
- SMIL: Un format per estructurar multimèdia sincronitzada.

També han hagut moltes iniciatives per treure profit de XML a sistemes relacionats amb la música.

La majoria són iniciatives relacionades amb la notació musical (MusicXML[51], MusiXML[34], MML[72], SMDL[57], MusicML[77]...), la generació de música (FlowML[68],Metrix[21]...), sincronització de multimèdia (MuTaTeD[22], SMIL[24], SMDL[57]...) i bases de dades multimèdia (MusicBrainz[59]).

4.10 MPEG-7

Les tecnologies digitals han facilitat l'accés massiu als mitjans de producció de multimèdia i l'oferta de contingut multimèdia disponible s'ha disparat. Tanta oferta, des del punt de vista dels usuaris, dificulta l'accés als continguts requerits perquè afegeix molt soroll. Des del punt de vista del productor, redueix la probabilitat ser trobat per un receptor potencial.

Això fa necessària la implementació de motors de cerca i indexació.

Les tecnologies de cerca més utilitzades a la xarxa es basen en indexació de text, amb la qual cosa la cerca ha de refiar-se de que el text que acompanya als continguts multimèdia realment es refereix a aquests continguts i els descriuen fidelment, quan, de fet, és molt possible que sigui un text arbitrari sense cap o amb poca relació amb els continguts multimèdia.

S'en desprén la necessitat de descriure la multimèdia d'una forma objectiva i ajustada al tipus de recurs per que permeti la seva indexació i cerca basant-se en característiques del propi contingut.

MPEG-7, anomenat formalment *Multimèdia content description interface*, és un estàndard que va néixer amb l'objectiu de cobrir aquesta necessitat.

A diferència dels estàndards descrits en els anteriors apartats, MPEG-7 no està promoguda per el W3C sinó que està promoguda pel Motion Pictures Expert Group, el mateix grup que va promoure els estàndards MPEG-1, MPEG-2 i MPEG-4. Els dos primer es refereixen a la compressió de dades multimèdia adoptant una perspectiva lineal de la informació, veient-la com un flux continu. MPEG-4 va més enllà i proposa una perspectiva estructurada i orientada a objectes, on els objectes, naturals i sintètics, es combinen per composar una escena amb la que l'usuari pot interaccionar.

MPEG-7 no s'adreça, com els seus predecessors, a la representació del contingut (dades) sinó que s'adreça a la descripció del contingut o metadades (dades sobre les dades). És un desplaçament en els esforços del MPEG des de la representació del contingut cap a la gestió d'aquest.

MPEG-7 es centra en el format dels descriptors, però, com als anteriors estàndards MPEG, no s'entra a normalitzar els algorismes per generar-los, només es fan suggerències. És per això, que MPEG-7 és molt important per CLAM. Si CLAM pot interaccionar amb MPEG-7, es podran desenvolupar amb CLAM algorismes de generació o processament de descripcions de continguts.

Els principals objectius[62] que ha assolit MPEG-7 són:

Base àmplia d'aplicacións: No està lligat a un tipus concret d'aplicacions com ara aplicacions d'streaming, de temps real, d'internet, de telefonia...

Diversitat de lligams amb el contigut descrit: No hi ha restriccions en la forma en que la descripció i el contingut estan lligats. Poden ser independents o poden estar integrats.

Independent del mitjà físic: El contingut referit pot estar físicament en qualsevol suport: disc dur, CD, internet, paper imprés...

Basat en objectes: El contingut està compost d'objectes amb les seves descripcions independents que poden ser accedides de forma individual.

Independent del format: No hi ha cap tipus de restricció en el format en el que es representa el contingut referit.

Múltiples nivells d'abstracció: La descripció pot fer-se situant-se en diferents nivells d'abstracció (de nivell físic a conceptual) i amb diferents granularitats (més o menys detall).

Extensibilitat: Ofereix una manera estàndard de fer extensions, doncs, seria molt poc realista pretendre abastar tot possibles descriptors o esquemes de descripció que puguin ser necessaris per qualsevol tipus d'aplicació.

Hi ha cinc mòduls importants a l'estàndard:

- Audio: descripció exclusiva d'àudio.
- Visual: descripció exclusiva d'imatge i vídeo.
- Multimèdia Description Schemes: descripció de la integració d'àudio i vídeo.
- Description Definition Language: definició d'extensions als descriptors.
- System: integració amb l'entorn, suports, formats...

MPEG-7 fa servir intensivament les tecnologies XML. Les descripcións del contingut multimèdia es realitzen en dos formats: Un format textual (TeM) que és, de fet, un arxiu XML, i, un altre format binari (BiM).

El format textual és un format XML que conté els descriptors. Aquest format, està especificat per un esquema de descriptors, anomenat DDL (*Description Definition Language*)[17] que és, de fet un XML-Schema amb algunes extensions que supleixen algunes carències dels tipus bàsics oferits per XMLSchema:

- Vectors multidimensió podent especificar les mides de cada dimensió.
- Poder especificar en la instanciació de l'esquema la mida de cada dimensió.
- Suport per punts temporals i duracions.

Amb DDL es defineix el format dels descriptors que estan considerats a l'estàntard, però també ofereix una forma d'ampliar aquest esquema a les necessitats d'una aplicació concreta fent servir les possibilitats d'extensió d'XML-Schema.

El format binari (BiM)[52] és especialment interessant perquè, de fet, tot i ser binari, manté les característiques XML i el podem tractar conceptualment com a tal. La codificació que en fan dels descriptors es basa en el fet que, donat un esquema, es pot generar un algorisme de traducció de l'esquema a un format binari compacte i viceversa.

Restringint un document XML amb un esquema podem saber dos coses:

- Per un costat, es coneix l'estructura que pot tenir cada element un cop especificat el seu tipus. S'en desprén que podem identificar els elements i els atributs amb seqüències binàries que els distingeixen entre les alternatives, ara acotades.
- Per altre costat, com es coneix el tipus concret de dada i la seva semàntica, es pot fer una codificació binària adaptada. Per exemple, codificant en binari, directament, les dades numèriques o o altre tipus concrets de dada com ara les marques de temps.

A més aquest format té l'avantatge de que, a diferència d'altres solucions proposades, basades en compressió del text, es conserven identificats els elements estructurals i podem fer-ne una extracció independent.

4.11 Conclusions

En el decurs d'aquest capítol he fet una descripció crítica de les diferents tecnologies relacionades amb l'estàndard XML. Aquest anàlisi és el resultat d'un estudi previ al disseny. Ha estat útil per contextualitzar i ha estat font d'idees.

XML és una matriu molt expressiva per formats de dades estructurats que facilita la interacció entre sistemes.

Aquest format es fa servir també a MPEG-7 per descriure el contingut multimèdia. Això reforça l'opció de donar suport a CLAM pel format XML. Tot i que CLAM no està restringit a generar MPEG-7 estricte, es poden crear un script XSLT que en faci fàcilment la traducció.

XML és un format basat en text que pot ser llegit i editat per humans. Malgrat que això sigui bo en molts casos, implica alguns problemes d'eficiència quan es treballa amb continguts típicament binaris. A l'apartat 4.1.4, he proposat algunes solucions parcial per aquests problemes, que es poden arribar a fer servir, però, com s'ha vist a l'apartat 4.10, MPEG-7 proposa una solució binària molt millor basada en XML-Schema que ens permet continuar pensant en la viabilitat d'una codificació purament XML.

Capítol 5

CLAM - C++ Library for Audio and Music

CLAM és el projecte matriu del present projecte de final de carrera. Aquest capítol de la memòria explica alguns aspectes de CLAM que serviran per establir el context en el que es mou el projecte. Cal dir que, si bé he format part activa en l'anàlisis, el disseny i la implementació de CLAM i en molts altres aspectes que s'expliquen en aquest capítol, aquests no són la part central del projecte, sinó el que s'explica al capítol 7.

Al final del capítol (apartat 5.9) faig un anàlisis dels rols que pot adoptar XML en el sistema, és a dir, quines són les utilitats que té el present projecte.

5.1 Introducció

CLAM (C++ Library for Audio and Music) és un projecte que s'està desenvolupant dins del Grup de Tecnologia Musical de l'Institut Universitari de l'Audiovisual (IUA-MTG) a la Universitat Pompeu Fabra.

Al mateix temps CLAM forma part del projecte europeu IST Agnula[1] l'objectiu del

qual és produir dos distribucions Linux orientades a la producció multimèdia, una basada en Debian (DeMuDI) i una altra basada en RedHad (ReHMuDi). La primera beta d'aquestes distribucions està planificada per al novembre de 2002.

CLAM és una biblioteca orientada a objectes en C++ amb l'objectiu inicial de que el codi dels diferents projectes relacionats amb àudio i tecnologia musical que es fan al grup de recerca es pugui re-usar i convinar fàcilment, dotant als projectes de:

- una infraestructura comuna per al desenvolupament,
- una manera uniforme de fer les coses, i,
- un repositori organitzat de desenvolupaments ja realitzats i directament usables.

Després d'entrar a formar part del projecte Agnula, CLAM ha passat a distribuir-se com a Software Lliure[7] sota llicència GPL[9], i els seus objectius pel que fa a l'àmbit, han passat, d'estar afitats als projectes del MTG, a abastar qualsevol projecte de la comunitat open source.

CLAM representa una reimplementació des de zero d'un conjunt d'algorismes i classes que existien amb anterioritat i que havien estat la base dels desenvolupaments del MTG fins fa poc. El projecte CLAM va començar a desenvolupar-se activament al setembre de 2000 tot i que, al començament, a falta de nom, era designat simplement com les classes del MTG (MTG-Classes).

Fins ara s'han fet entregues internes, adreçades als altres projectes del MTG. Més endavant, al Juliol de 2002 es farà una primera entrega pública i, finalment, al Novembre de 2002 es farà la primera entrega conjunta amb la beta d'Agnula.

5.2 Objectius de CLAM

Donat que es vol construir una base comuna per el desenvolupament dels diferents projectes del grup, no només és útil cobrir les necessitats purament de processament

sinó que s'intenta donar una solució re-usable a altres camps de les aplicacions com ara l'entrada i sortida, la representació gràfica, el multi-fil, i la interacció amb perifèrics musicals.

Es pot dir que CLAM no només pretén ser una biblioteca de processament sinó també un framework complert per aplicacions de processat d'àudio i música.

Tot i així, la part central de processament s'ha mantingut independent de la resta perquè es pugui integrar dintre d'una aplicació que ja doni suport a aquests aspectes accessoris.

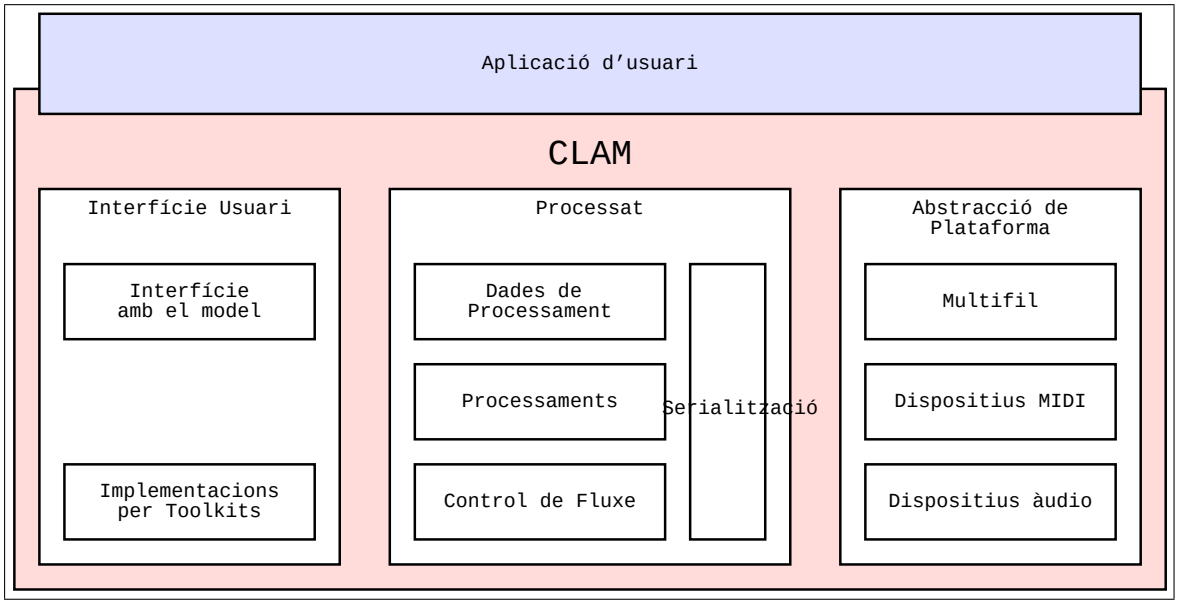

Figura 5.1: Diagrama de mòduls de CLAM

Hi ha projectes del MTG que funcionen en una o diverses arquitectures. CLAM ha de funcionar en totes elles i, a més, ha de fer possible que un projecte desenvolupat en una plataforma funcioni sense gaires problemes en una altre. Les plataformes que es volen suportar són Windows, Linux i Machintosh. Actualment les dues primeres es suporten completament i la darrera es suporta en gran part a falta de portar algunes interfícies hardware.

5.3 Arquitectura de processament

CLAM fa servir bàsicament una arquitectura de processament modular. El sistema és una xarxa d'elements de procés o mòduls pels quals van passant les dades com a una cadena de muntatge per obtenir el producte final.

Així doncs, la clau per definir l'arquitectura està en definir:

- un encapsulament pels algorismes,
- un format per les dades, i
- els patrons de flux.

Aquesta arquitectura és molt semblant a la de moltes altres biblioteques de processament. La diferència, però, radica en la versatilitat que la biblioteca requereix per donar cabuda als problemes que es voldran resoldre.

5.4 Processos

Tipus d'algorismes

Els projectes que es fan al grup de recerca, i que CLAM ha de suportar, són sobretot projectes que treballen en la transició entre els dominis sintàctics i semàntics de l'àudio. És a dir, en un extrem hi ha el processament de senyal, a baix nivell, basat en propietats físiques, i a l'altre extrem tenim un processament de continguts, a alt nivell, que està basat en la interpretació cognitiva d'aquestes propietats.

CLAM permet que aquests dos extrems dialoguin donant suport a les diferents capes d'abstracció que hi ha entre un i altre extrem.

Encapsulament

CLAM encapsula els algorismes de processament en objectes de processament o processos (`Processing`). Aquests objectes ens donen la interfície per executar explícitament un pas de l'algorisme encapsulat.

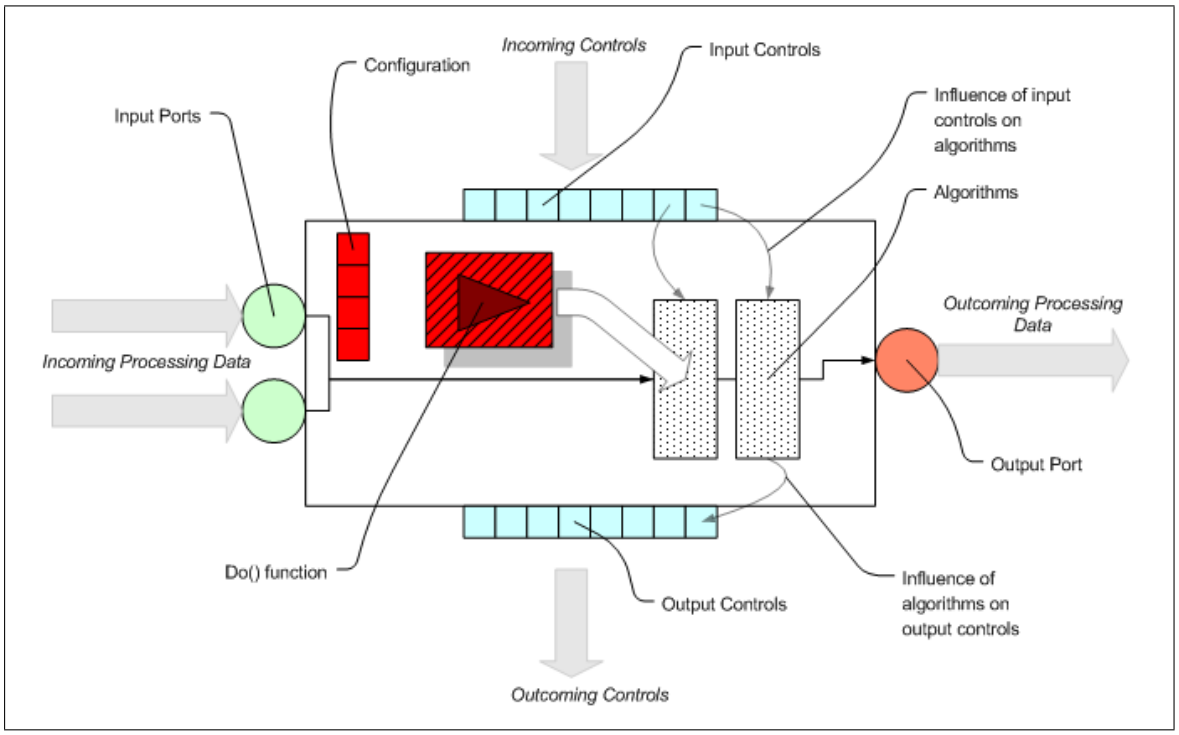

Figura 5.2: Representació d'un element de procés

A cada pas de l'algorisme, els processos consumeixen dades d'entrada, i, com a resultat del processament, en produeixen de sortida.

La possibilitat d'un procés per consumir o produir dades s'abstreu amb el concepte de port. Els ports d'un procés ens permeten obtenir informació de quines necessitats de dades té.

Parametrització i control

Els processos, abans de començar la seva execució, poden rebre una configuració inicial per inicialitzar-se i construir les seves estructures internes segons els paràmetres d'aquesta configuració.

Un cop en estat d'execució, hi ha certs paràmetres, els controls, que es poden modificar amb events que altres processos envien des dels seus controls de sortida.

Escalabilitat

També podem agrupar un seguit de processos i abstreure-los en conjunt com a un únic procés que podem fer servir en altres sistemes més grans. D'aquesta manera, el sistema resulta escalable per a l'usuari a nivell d'ús, atès que pot ignorar, si vol, la complexitat del que hi ha a dins d'aquest procés.

5.5 Dades de processament

Representant el senyal

Els processos es nodreixen de dades. Les dades viatgen entre els diferents processos en forma d'unitats o tokens. A la majoria d'entorns existents que permeten dissenyar sistemes de processament, aquestes dades són fragments del senyal d'àudio en forma temporal o freqüencial.

CLAM dóna suport tant al processament temporal com al freqüencial. El processament temporal es fa, no pas agafant com a token bàsic la mostra sinó que es fa servir el buffer d'àudio. Un buffer d'àudio és un encapsulament d'un conjunt de mostres contígües a memòria i certa informació temporal.

La informació temporal ajuda a manegar finestres d'àudio no adjacents, desordenades o superposades en el temps i mantenir el sincronisme entre els diferents fluxes de

dades.

Pel que fa al processament freqüencial, CLAM dóna suport a les diverses representacions que hi ha en aquest domini. Cada processat s'hi avé amb una representació concreta i podem trobar el punt òptim entre fer servir la representació òptima per a cada processament o minimitzar el nombre de conversions.

També es dóna suport a representacions del senyal de granularitat major (el segment) o a representacions del senyal alternatives com ara el modelat espectral (harmònics + soroll)[69].

Descrivint el senyal

Els requeriments de CLAM, però, exigeixen que les dades puguin representar més coses que no pas el senyal físic d'àudio. Necessitem que els processos puguin intercanviar construccions amb resultats d'anàlisis intermedis i descriptors amb el contingut sintàctic i semàntic extret del senyal.

Els resultats d'anàlisis són per natura informacions estructurades que cal representar amb objectes molt més complexos que no pas un espectre o una tira de mostres, però, cal gestionar-los d'una forma semblant a aquests, mantenint una certa uniformitat per facilitar el control de flux.

5.6 Patrons de flux

El flux en un sistema de processament determina l'ordre d'execució dels elements de processament i la forma en que les dades es generen, passen d'un element a un altre i es modifiquen.

CLAM permetrà dos modes d'operació:

- Un *mode supervisat* per CLAM on el flux és totalment gestionat per la llibreria.

- Un de no supervisat, on el programador és qui nodreix als elements de procés amb les dades.

En la primera fase de CLAM, només el mode no supervisat està implementat. El mode supervisat està pensat per molt més endavant.

Si bé el disseny de CLAM pot ser complex degut a la heterogeneïtat de les dades que ha de gestionar, ho és molt més pels requeriments que té el flux d'aquestes dades.

El flux més simple que cal suportar és aquell en que tots els processos consumeixen i produeixen dades amb el mateix ritme. En aquest cas, simplement cal executar els processos en ordre tal que cadascun s'executi amb dades disponibles.

A part d'aquest cas simple, CLAM ha de donar suport a molts altres patrons de flux que s'expliquen a continuació.

Re-alimentacions

El patró anterior necessita algunes modificacions quan introduïm alguna re-alimentació, que són molt comunes als sistemes de processament. Les re-alimentacions no disposen de dades per les primeres execucions i cal donar-ne dades per defecte.

Les realimentacions també compliquen els algorismes per determinar l'ordre d'execució a partir de la topologia. Fins i tot, hi ha topologies que poden arribar a esdevenir ambígües com, per exemple, la que es mostra a la figura 5.3. Els tres casos es basen en la mateixa topologia, però, s'obtè resultats diferents segons es consideri quina conexió és la realimentació.

Flux multirate

El següent refinament que ens podem trobar és el **flux multirate**. Es dóna quan tots els elements de procés no produeixen i consumeixen dades amb la mateixa proporció a cada pas d'execució. Per assegurar que a cada pas hi hagi dades, cal executar els

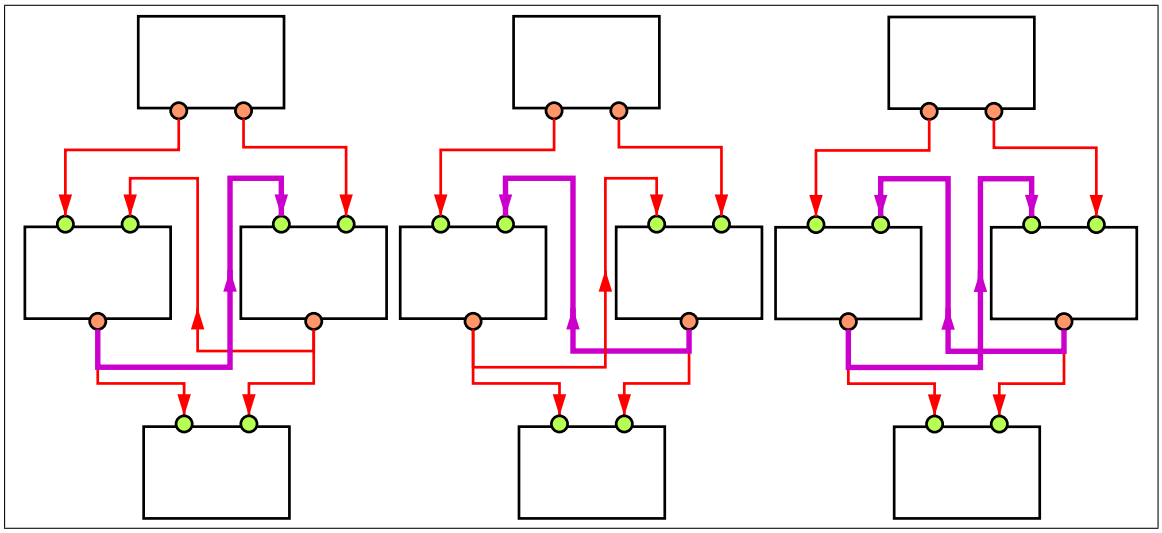

Figura 5.3: Topologia amb re-alimentació ambígua

passos de cada procés amb diferent freqüència.

Flux variable

El cas extrem en aquest sentit és aquell en el que el nombre de dades consumides o produïdes varia d'uns passos a uns altres. El **flux variable** obliga a mantenir certa intel·ligència en temps d'execució que, si només haguéssim de fer servir flux fix, seria necessària només en temps de configuració.

Inplace processing

En els casos anteriors es considerava que les dades d'entrada només es llegeixen i no es modifiquen. Les dades de sortida són dades verges a les que el procés dóna un valor. Això va bé perquè es poden fer moltes suposicions de cara al flux, però no sempre és la millor opció.

Quan les dades són estructures complexes, i el processament consisteix en obtenir l'entrada modificada, aquest tipus de flux esdevé molt ineficient perquè a cada execució

de l'algorisme cal tornar a crear l'estructura de la dada.

La solució és fer que la dada d'entrada i sortida sigui la mateixa modificant-la directament. Això ho anomenem *inplace processing*.

El *inplace processing* complica relativament el flux atès que cal controlar l'accés a la dada i modelar les dependències de dades que es generen, de forma molt semblant a com ho faríem als registres d'un processador. Hi ha un procés que genera la dada, un que la llegeix, un altre que la modifica i un altre que llegeix la modificació. Cal ordenar l'execució d'aquests processos per a no crear inconsistències.

Recàlculs

Els recàlculs es fan servir normalment per corregir les suposicions que es fan durant els transitoris d'un so.

Per exemple, si volem fer un processament sobre el so d'un instrument que es basa en el pitch de la nota, probablement en els transitoris, quan comença una nota, farem suposicions equivocades del pitch i els càlculs seran erronis.

Per obtenir un millor resultat, convé refer els càlculs un cop s'ha detectat l'error.

Els recàlculs són complexos de gestionar amb un control de flux genèric però cal implementar-los perquè, en donar bons resultats, són comuns als sistemes que CLAM ha d'implementar.

Flux asíncron (controls)

El flux asíncron és aquell que no es produeix de forma contínua sinó en forma d'events.

Els processos es poden comunicar informació asíncrona entre ells, però, no ho fan de la mateixa manera que es comuniquen les dades sinó fent servir el mecanisme de controls que hem comentat abans.

5.7 Latència i eficiència

Les aplicacions d'streaming són aquelles en que les dades a processar s'ofereixen a un ritme continu en el temps i no es pot suposar una durada determinada del flux. L'streaming imposa també uns requeriments d'eficiència mínims per a ser útil: El processament ha de processar les dades amb la mateixa velocitat amb que arriben.

Per suavitzar aquests requeriments d'eficiència es fan servir buffers grans tant a l'entrada com a la sortida. Aquests buffers serveixen per compensar els pics de processament i els canvis de context prolongats. Amb els buffers, els requeriments d'eficiència són només en mitjana i els pics màxims poden ser molt més alts si no s'allarguen massa. Els buffers, però, incrementen el temps que passa entre que la dada arriba i es processa el resultat, la latència.

Les aplicacions en temps real són molt semblants a les aplicacions d'streaming, en el sentit que necessiten processar les dades amb la mateixa velocitat amb que arriben dades a l'entrada. Però, en aquest tipus d'aplicacions pren importància la el temps de resposta respecte als canvis de l'entrada. És per això, que en aquestes aplicacions no es pot fer servir buffers gaire grans, donat que el temps de respota del sistema depén directament de la latència que s'afegeix.

La latència depèn en gran mesura del temps que triga la dada en processar-se, l'eficiència, però, tot i disposar d'un processador infinit seguiria existint una latència mínima que depèn, per exemple, de la quantitat de dades que es necessiten a l'entrada per acabar generant una sortida.

Per exemple, si estem fent processat espectral, hem d'esperar com a mínim a que entrin tantes mostres com la mida del frame per poder generar una sortida, i, aquesta latència hi serà independentment de la velocitat del processador. En dissenyar el control de flux cal anar en compte donat que els buffers de dades que necessitem pel control de flux poden afegir latència mínima extra a la propia de l'algorisme de processat.

Tot i així, quan l'eficiència no és suficient per arribar a la velocitat d'entrada de

dades, a vegades convé afegir latència mínima si amb això guanyem en eficiència. Per exemple, un processament mostra a mostra té menor latència mínima que un processat en blocs, però, un processat en blocs aprofita més la memòria cau i és més ràpid. Potser la latència mínima que afegim queda absorbida per un altre element que també necessita un processament en blocs.

La latència afegida pel processament, també pot ser absorvida per la latència que afegeix el subsistema d'àudio: la llibreria d'àudio, el sistema operatiu, els drivers i la targeta. L'equip de CLAM ha invertit molt de temps en l'optimització de la latència al subsistema d'àudio de les diferents plataformes.

5.8 Casos d'ús de CLAM

Aquest apartat explica alguns dels casos d'ús de la llibreria. En llibreries amb el nivell de generalitat de CLAM és difícil avaluar quan algun aspecte del seu disseny és útil o necessari. Així, aquests casos d'ús ens serviran per projectar sobre casos reals les abstraccions de disseny que en fem a la memòria.

La majoria de casos d'ús són aplicacions que es varen desenvolupar directament amb CLAM. D'altres són aplicacions existents amb anterioritat que s'hi varen portar. El segon cas ens ha estat molt útil per poder establir comparatives amb les implementacions anteriors.

Tot i que algunes aplicacions es varen fer expressament per testar la llibreria, bona part projectes independents que es varen desenvolupar de forma paral·lela a CLAM.

Aquests projectes paral·lels han patit els canvis de CLAM durant la seva maduració, però, tenir tantes aplicacions funcionant amb CLAM durant el desenvolupament ens ha estat molt útil: ha enfortir la validesa d'algunes propostes que es feien i ha detectat les febleses d'altres.

De fet, a les aplicacions, tot i la molèstia dels canvis, els hi compensaven les millores que feien en eficiència i funcionalitats noves.

També varen servir per centrar els dissenys sovint massa generals en casos concrets i donar resultats tangibles ja abans del final del projecte.

Spectral Delay

La primera aplicació que vam desenvolupar amb CLAM va ser un retard espectral. Vam escollir aquest efecte perquè, incloïa alguns dels elements més comuns als sistemes objectiu, tot i mantenir certa senzillesa.

Aquests elements són l'enfinestrat (un producte de buffers d'àudio en domini temporal), la transformada ràpida de Fourier i la seva inversa per passar del domini temporal al espectral, un producte d'espectres per filtrar cadascuna de les bandes, un generador d'espectres en format BPF (funcions definides per interpolació de punts) per generar els perfils dels filtres de cadascuna de les bandes i un sumador amb ensolapament per sumar les bandes tenint en compte l'enfinestrat i obtenir el resultat.

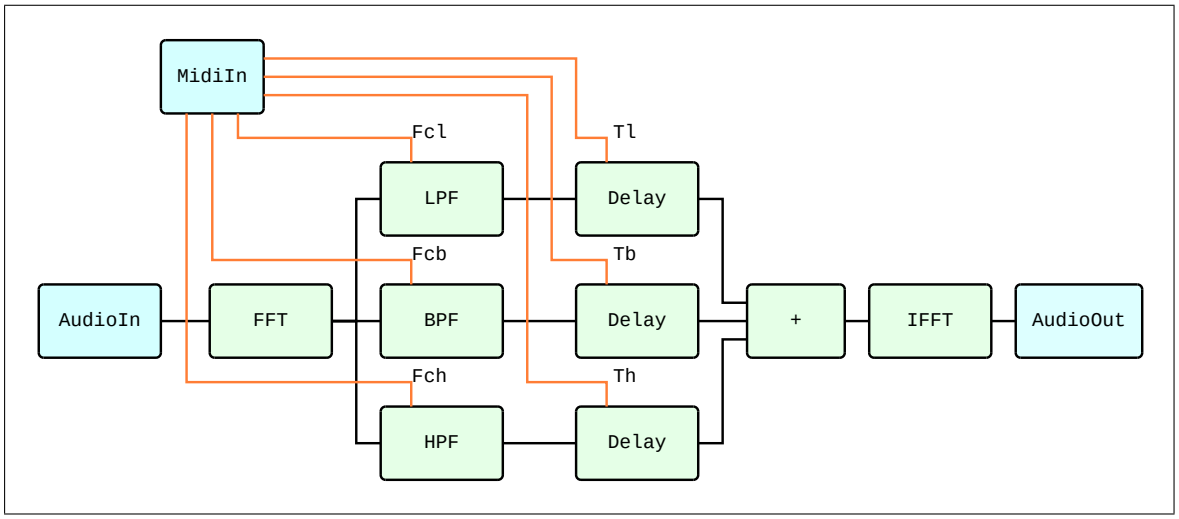

Figura 5.4: Esquema de l'Spectral Delay

A més, a aquest senzill esquema se li poden anar afegint moltes de les funcionalitats d'ús general com ara les diferents fonts i destí de dades (formats diversos de fitxers, targeta de so, streaming...), interfície gràfica, manipulació de controls des de l'interfície gràfica o des d'un dispositiu MIDI...

SMS

Spectral Modeling Synthesis[69] és un algorisme d'anàlisis i resíntesis en el que es basen molts dels desenvolupaments del MTG. L'algorisme extreu del senyal una representació que permet algunes manipulacions més orientades al contingut.

La representació alternativa s'extreu mitjançant l'anàlisi espectral del senyal, separant la part harmònica del soroll i, extraient-ne alguns descriptors. Després de manipular aquesta representació es pot tornar a resintetitzar obtenint-ne el resultat final.

Rappid

Rappid és un sistema de temps real basat en CLAM que està pensat per fer-se servir en un espectacle musical en viu. La idea general és que dos intèrprets musicals toquin cadascun un instrument (una viola i un arpa) i un tercer intèrpret, amb l'ordinador, generi un instrument virtual tot intermodulant ambdós senyals amb paràmetres de control extrets dels mateixos dos senyals. El tercer intèrpret controla quins són els paràmetres de control de cada senyal i de quina manera n'afecten al resultat.

És una aplicació real que es vol fer servir per a una obra musical d'en Gabriel Brncic.

El principal tret d'aquest sistema és el seu àmbit d'ús. No és una eina d'estudi, és una eina de directe que requereix poca latència per poder ser controlada efectivament. El tercer intèrpret es podria arribar a programar o deixar fix, però, l'entrada dels altres dos sempre és en viu.

El sistema de processament ha d'extreure paràmetres de control de les senyals. Què són aquests paràmetres de control? Una envolupant de la senyal, les dades del timbre, descriptors de melodia... Està clar que aquests paràmetres de control poden tenir molt a veure amb els descriptors que puguem extreure i la capacitat d'aplicar aquests descriptors per modificar l'altre senyal.

Els sistema també necessita una interacció en temps real amb el tercer intèrpret de forma que necessitem introduir controls que aquest pugui manegar, bé amb una

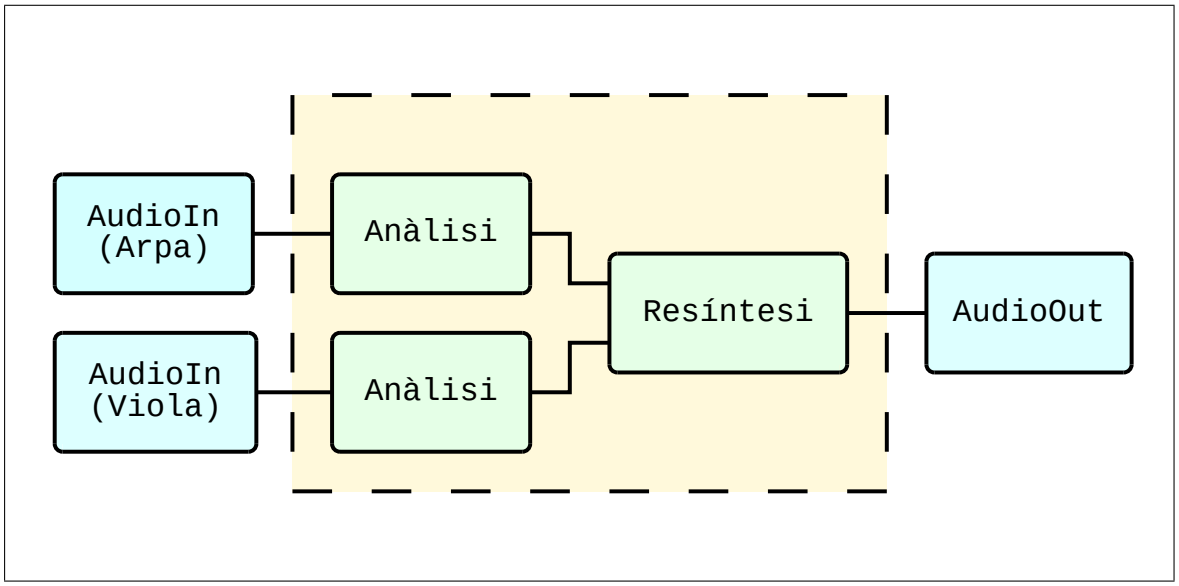

Figura 5.5: Esquema de mòduls de Rappid

interfície gràfica a l'ordinador, bé amb dispositius MIDI o semblants.

A més, seria interessant que el sistema pogués fer-se configurable, de forma que l'algorisme que fa la intermodulació es pugui escollir segons la peça musical.

5.9 Oportunitats d'XML dins de CLAM

Un cop descrita la llibreria CLAM i els seus casos d'ús, en aquest apartat s'analitzen els àmbits de CLAM on és útil oferir suport XML.

5.9.1 Passivació i activació de dades

El primer àmbit clar, és la representació de les dades de processament a disc. Si fos un entorn de processament de senyal convencional no tindria gaire sentit representar un senyal digital en un format de text. XML es massa redundant i un format binari seria molt més convenient per aquest tipus de dades.

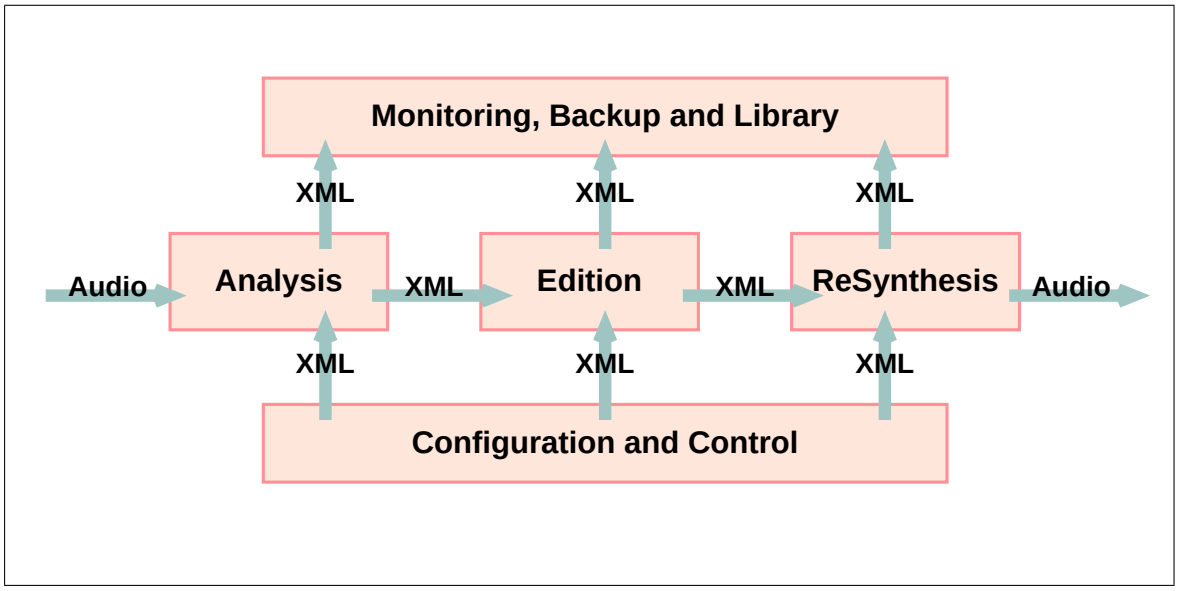

Figura 5.6: Papers que pot adoptar XML a CLAM

Malgrat tot CLAM manega tipus de dades molt més heterogenis i la representació en XML comença a prendre sentit quan parlem de descriptors i de resultats d'anàlisis estructurats.

5.9.2 Interfície entre sistemes

Altre motiu pel que XML és interessant és la seva capacitat de fer d'interfície entre sistemes diferents. Ja poden ser sistemes en diferents plataformes, diferents proveïdors o amb funcionalitats complementàries, XML afavoreix aquest intercanvi.

Per un costat, XML és base comuna de formats diversos i existeixen llibreries que llegeixen aquesta base comuna sent molt fàcil llegir un format XML qualsevol a partir del model de document a memòria i una definició d'aquest format.

XML és només un patró de format, no pas un format concret. Una mateixa informació pot ser representada en una infinitat de formats XML diferents. Afortunadament, si no volem adaptar cada aplicació perquè pugui llegir cada format, sempre es pot fer

servir XSLT. Com s'explica a l'apartat 4.6, XSLT és un llenguatge de transformació de documents XML que està pensat especialment per a fer traduccions entre diferents formats XML.

5.9.3 MPEG-7

Hi ha un altre element que fa que el suport de XML a CLAM sigui valuós com a interfície entre sistemes.

Recentment s'ha alliberat la primera versió de l'estàndard MPEG-7[40] que estandarditza com descriure els continguts multimèdia. Aquesta descripció dels continguts inclou interpretació del significat del contingut i esta pensada per poder ser tramesa o accedida per agents electrònics.

Per exemple, en el cas de l'àudio, i concretament àudio representant continguts musicals, exemples d'aquests descriptors podrien ser l'autor, l'intèrpret, el títol, la lletra, la partitura, la tonalitat, el registre, les relacions amb altres suports multimèdia, les condicions d'enregistrament...

Evidentment, molts d'aquests descriptors multimèdia hauran de ser creats a mà per un operador, però, CLAM és ideal per obtenir-ne de forma automatitzada gran part d'aquests descriptors.

MPEG-7 està molt lligat a XML perquè el format escollit per emmagatzemar els descriptors és un format XML. Així doncs, donant la possibilitat de generar i llegir XML estem possibilitant la compatibilitat amb MPEG-7 ja sigui directament, ajustant a l'estàndard el format generat per CLAM, o bé fent simples transformacions XSLT.

5.9.4 Configuració

XML també és un format ideal per a fitxers de configuració. Precisament ho és perquè és llegible i modificable pels humans. A més, la configuració en XML es pot organitzar de forma estructurada.

Què es pot configurar a CLAM amb fitxers XML? Principalment, els paràmetres configurables dels mòduls de processat. També les opcions pròpies de l'aplicació.

Si construir un objecte estructurat que contingui la informació de configuració podrem carregar-ho a partir d'un document XML, consultar els valors, modificar-los i guardar-los a un suport físic.

5.9.5 Especificació de sistemes

Un sistema de processament en CLAM és una xarxa d'elements de processament interconectats entre ells. Si ampliem l'ús d'XML per poder representar aquestes xarxes, podrem desar i obrir en un suport físic els dissenys.

Com s'ha comentat prèviament, CLAM tenia certes funcionalitats relatives a l'escalabilitat en el disseny que permetien abstraure una xarxa com a un únic element de procés dintre d'una xarxa més gran. Així doncs podem incorporar una xarxa completa desada anteriorment com a document XML dintre d'un disseny nou.

Un cop dissenyada una xarxa, la podrem reutilitzar en qualsevol altre disseny facilitant així un disseny incremental dels sistemes.

A més, CLAM podrà proporcionar, a més dels elements de procés programats en C++, una bateria d'elements de procés no compilats en format XML.

El llistat 5.1 és un possible format XML per a representar una xarxa.

Tot i que aquestes xarxes d'elements no serien tan òptimes com ara les compilades, donat que són més fàcils de construir pels usuaris, ampliarien el nombre d'algorismes implementats. A més, sembla molt factible generar el codi C++ a partir de la descripció de la xarxa en XML.

```
<process-type
    id="spectral.fft.useless-step-fft"
    base-name="fft"
    icon="default.png"
>
    <description>La descripció de la xarxa va aquí</description>
    <process id="fft1" type="spectral.fft.my-fft-implementation">
        <!-- Process configuration goes here -->
    </process>
    <process id="ifft" type="spectral.ifft.my-ifft-implementation">
        <!-- Process configuration goes here -->
    </process>
    <process id="fft2" type="spectral.fft.my-fft-implementation">
        <!-- Process configuration goes here -->
    </process>
    <link>
        <output owner="fft1" id="spectral-output" />
        <input owner="ifft1" id="spectral-input" />
    </link>
    <link>
        <output owner="ifft1" id="timedomain-output" />
        <input owner="fft2" id="timedomain-input" />
    </link>
    <export-input name="timedomain-input">
        <description> L'entrada </description>
        <input owner="fft1" id="timedomain-input" />
    </export-input>
    <export-output name="timedomain-input">
        <description> La sortida </description>
        <output owner="fft2" id="spectral-output" />
    </export-output>
    <export-in-control name="Window Size">
        <description> La mida del frame </description>
        <in-control owner="fft1" id="Window Size" />
        <in-control owner="fft2" id="Window Size" />
        <in-control owner="ifft1" id="Window Size" />
    </export-in-control>
    <export-out-control name="Window Size">
        <description> La <b>mida</b> del frame </description>
        <out-control owner="fft1" id="Window Size" />
    </export-out-control>
</process-type>
```

Llistat 5.1: Possible format per representar xarxes

5.9.6 Resguard de l'estat d'un procés

Les aplicacions que no fan el processat d'àudio en temps real, sovint fan càlculs que es prolonguen molt en el temps. Un temps de processat que es pot perdre si el sistema cau o si es necessita interrompre el procés deliberadament.

La funcionalitat d'XML permetria d'una forma molt directa implementar un mecanisme de passivació per emmagatzemar l'estat intermedi d'un sistema de processament.

Això hauria d'incloure l'estat intern dels element de procés, si és que en tenen, i l'estat de les estructures de flux com ara el contingut dels buffers que comuniquen els processos.

Un sistema com aquest permetria interrompre un procés llarg quan, per exemple, el sistema esta massa carregat o necessita re-iniciar-se, per posteriorment reprendre'l.

Passivacions de resguard periòdiques i automàtiques de l'estat de computació en XML pot evitar el malbaratament del temps de computació invertit si el sistema pateix una interrupció sobtada.

5.9.7 Suport al programador

Es pot aprofitar que XML és estructurat i pot representar l'estructura de les dades, per a que el programador, des del programa depurador si vol, pugui passivar en XML el contingut d'un objecte del programa i inspeccionar-ho.

De fet aquest ha estat un dels principals usos de XML durant el desenvolupament de CLAM, i segurament serà igualment explotat pels seus usuaris.

Capítol 6

Tecnologies pel disseny de llibreries

En el disseny de la llibreria CLAM, diverses tècniques que tenen a veure amb l'enginyeria del software han estat intensivament emprades. El suport XML participa d'aquest ús. En aquest capítol s'expliquen algunes d'aquestes eines, metodologies i tècniques.

6.1 Orientació a objectes

Avui en dia ignorar la programació orientada a objectes es com ignorar la programació estructurada o els llenguatges d'alt nivell. Qualsevol projecte d'una certa mida necessita beneficiar-se dels avantatges que ofereix la programació orientada a objectes [39]:

- Millora la comprensió del sistema: La modelització d'una aplicació amb objectes i relacions d'herència, composició i referència entre ells són eines cognitives naturals en els humans.
- Permet gestionar millor la complexitat: L'encapsulació de les dades darrera d'una interfície redueix el nombre de connexions entre les diferents parts de l'aplicació. També facilita el canvi reduint i localitzant els lligams entre les diferents parts del sistema.

- Dona extensibilitat i adaptabilitat al sistema: L'herència permet extendre o modificar els objectes existents i, gràcies al polimorfisme, es poden fer servir els nous objectes on hi havien els antics sense modificacions.
- Permet la reutilització de codi: Una classe és pot instanciar múltiples vegades. L'herència permet fer servir codi d'una classe en una altra.

Quan parlem d'orientació a objectes, cal tenir en compte amb quina visió es parla d'orientació a objectes. Cada llenguatge de programació en fa la seva visió. Anomenem idioma és cadascuna de les formes de treballar amb un mateix llenguatge. Fins i tot, a cada idioma la visió de l'orientació a objectes és diferent.

Més que cercar què és concretament l'orientació a objectes, Stroustrup, el pare de C++, dóna una visió més alliberadora[76]. Diu que no és important cercar la orientació a objectes, sinó cercar aquelles eines que siguin útils pels desenvolupadors siguin, no siguin o no s'estigui segur de si són, orientades a objectes.

6.2 Patrons de disseny

Molta de la feina del disseny de CLAM i d'aquest PFC ha vingut reforçada per l'anàlisi de patrons de disseny aplicats als diferents problemes trobats. El concepte de patró de disseny ha estat popularitzat arran del llibre d'Erich Gamma et al. [47], on s'estableix formalment el concepte i es defineixen amb exhaustivitat i de forma molt pràctica els patrons de disseny més comuns.

Un patró de disseny és una abstracció d'un problema freqüent que sovint es soluciona d'una manera similar. En fer l'analogia entre l'abstracció i el problema objectiu, s'identifica ràpidament els actors i les forces, i es pot avaluar si la solució oferida pel patró és acceptable pel cas de disseny.

Avança molt la feina d'avaluació, perquè, a l'especificació dels patrons de disseny, es remarca quins factors clau fan que un patró sigui l'idoni o no, tot i haver aconseguit fer l'analogia.

Els patrons de disseny són eines per transmetre i adquirir experiència en la resolució de problemes de disseny. Evidentment no tots els problemes són iguals, però, existeixen certes analogies entre ells que en abstreure-les esdevenen reaprofitables. No s'està parlant de reaprofitament de codi, sinó de reaprofitament de disseny.

Els patrons de disseny són també molt útils donat que ofereixen al grup de dissenyadors un llenguatge comú amb el que comunicar-se idees de disseny.

A [47] es descriuen alguns patrons de disseny en l'àmbit del disseny orientat a objectes. Aquests patrons de disseny els classifiquen en tres grans famílies:

- **Patrons de construcció:** Resolen com es creen els objectes.
- **Patrons d'estructuració:** Resolen com es combinen els objectes.
- **Patrons de comportament:** Resolen com modelar el comportament.

El concepte de patró de disseny també s'ha aplicat a altres àmbits de l'enginyeria del software, que no només al disseny orientat a objectes. Per exemple, hi ha bibliografia sobre patrons de disseny per la programació orientada a components o a agents distribuïts.

6.3 Programació genèrica

Si bé, la programació orientada a objectes o la programació orientada a components han promogut la reutilització de codi, aquesta arriba només fins a cert nivell.

El problema amb la programació orientada a objectes a la majoria de llenguatges és que es lliga massa els algorismes amb les dades que maneguen més del que els conceptes requereixen. És a dir, amb orientació a objectes pura no es pot programar un procediment independentment del tipus de dada que s'està manegant.

La programació genèrica permet programar algorismes centrant-se en els conceptes que representen les dades i no pas en els tipus concrets.

6.3.1 Programant amb conceptes

Igual que la programació orientada a objectes respecte a la programació estructurada, la programació genèrica no substitueix pas la programació orientada a objectes sinó que la expandeix.

La programació genèrica es una forma de programar que es basa només en els conceptes que representen els símbols. Un **concepte** és una família d'abstraccions que acompleixen un seguit de requeriments comuns, entenent per abstraccions tipus de dades o classes concretes.

Per exemple, es pot definir el concepte *ordenable* com aquell que agrupa totes les classes en les que es pot establir un ordre entre les seves instàncies mitjançant un operador <.

Hom pot dissenyar un algorisme tot suposant quelcom sobre aquest concepte. Per exemple, dissenyem un algorisme per escollir el major element de dos donats. Aquest algorisme serà aplicable a un parell d'objectes de qualsevol classe que pertanyi al concepte *ordenable*.

6.3.2 Parts d'un concepte

Parlant d'una forma més formal, un concepte és un conjunt de requeriments que un tipus concret pot satisfer. Aquests requeriments s'adrecen a quatre aspectes:

Conjunt d'expressions vàlides: Aquestes expressions són les que s'han de poder fer servir amb els objectes del tipus que satisfà aquest concepte.

Tipus associats: Quan un tipus modela un concepte, hi ha un seguit de tipus associats que adopten rols concrets segons el concepte. El concepte pot exigir que aquests tipus associats també modelin altres conceptes.

Per exemple, creem el concepte *contenidor ordenat* amb dos rols associats: *elementBasic* i *iterador* que requereixen respectivament acomplir els conceptes

ordenable i *iterador*.

Quan un vector de nombres reals modela el concepte *contenidor ordenat*, automàticament els nombres reals modelen el concepte *ordenable* i l'iterador que ens proporciona el vector modela el concepte *iterador*.

Invariants: Els invariants tenen més a veure amb la semàntica de les expressions vàlides i es modelen amb precondicions i postcondicions.

Els invariants necessiten un llenguatge formal per poder-se expressar i verificar.

Garanties de complexitat: De cara a la usabilitat es garanteix per a cada operació una complexitat asimptòtica màxima.

Aquest darrer grup de requeriments és difícil de comprovar. De fet només s'afegeix com a part de la documentació per que els usuaris d'una classe coneguin que aquests requeriments es compleixen.

6.3.3 Implementacions

Els diferents llenguatges de programació han arribat a diferents aproximacions per implementar parcialment els conceptes. És a dir, fan servir diferents estratègies per re-aprofitar codi perquè funcioni per a un conjunt de classes que tenen alguna cosa en comú.

A la majoria de llenguatges orientats a objectes, el punt feble és la verificació de que els objectes genèrics estan dintre del concepte.

Smalltalk[50] tracta tots els objectes com a igual. Un programador d'Smalltalk pot passar qualsevol objecte com a paràmetre d'un mètode. És en fer servir aquest objecte dintre del mètode, quan li enviem un missatge, que es comprova si hi ha cap mètode associat amb el missatge enviat, és a dir, es comprova si pertany al concepte que s'esperava. Però, això es fa en temps d'execució i, si l'objecte no s'hi escau al concepte, el programa dóna un error d'execució.

En resum, en temps de compilació, Smalltalk no fa cap comprovació de tipus, la

qual cosa possibilita la programació genèrica, però, tampoc fa cap comprovació de concepte, la fa en temps d'execució. Això deriva en errors en temps d'execució no gaire controlables que es podrien haver detectat en compilació.

C++ sí té comprovació de tipus en temps de compilació, de fet, aquest fet impossibilitaria la programació genèrica si no fos per les classes abstractes i els *templates*.

Es poden implementar *conceptes* amb classes abstractes. Qualsevol instància d'una subclasse d'una classe abstracta es pot passar com a paràmetre enlloc on es demani un punter o una referència a la classe abstracta. L'ús d'aquesta instància és segur en temps d'execució donat que la instància de la subclasse ha de implementar la interfície de la classe abstracta. Això es comprova en temps de compilació.

Aquesta aproximació, però, té diversos problemes:

- Cal fer la relació del concepte amb la classe de forma explícita. Objectes que s'avinguin al concepte, però, no derivin de la classe abstracta no es poden fer servir.
- La definició de la classe esdevé complexa, donat que cal derivar de tots els conceptes als que pertany.
- Els conceptes poden tenir parts en comú i en aquest cas es complica la resolució de l'herència.
- Els tipus bàsics no poden encabir-se en cap concepte donat que no són classes que puguin derivar-se.
- La resolució dels mètodes virtuals es fa en temps d'execució i, encara que ara no provoca errors, si que no es pot resoldre de forma *inline*.

Java millora alguns d'aquests punts amb els interfaces. Els interfaces es fan servir només en el sentit que necessitem: especifiquen uns requeriments d'interfície. No hi ha cap implementació i la resolució de l'herència és més senzilla. Tot i així, també cal explicitar a la declaració de la classe quins conceptes s'hi avenen amb el tipus.

Els templates es varen introduir al llenguatge C++ precisament per poder fer programació genèrica. Són una aproximació al problema dels conceptes complementària respecte de l'ús de classes abstractes. Els templates són declaracions parametritzades, on els paràmetres de la declaració són tipus o constants senceres. En comptes de declarar una classe o una funció, estem declarant un motllo d'on sortiran diverses funcions o classes segons els valors.

En comptes de forçar la subclassificació i la implementació d'un interface complert, es pot programar un algorisme o una classe template tot suposant que el paràmetre amb el que es farà servir compleix l'interfície requerida.

Els templates aconsegueixen l'objectiu: implementar algorismes i estructures de dades d'una forma conceptual. Però, d'alguna manera aquesta implementació té alguns defectes que dificulten tant el desenvolupament d'una llibreria com el seu ús.

- Cap declaració és instanciada fins que no es fa servir. És molt difús saber si una declaració template es correcta fins que no es fa servir directament a nivell de símbol.
- Els errors de compilació de codi d'usuari, els dóna a dintre de la llibreria.
- Els símbols que representen declaracions template són costosos d'entendre.
- Els errors de compilació no són gaire clars si no estàs acostumat a fer servir templates de forma massiva.

A això cal afegir-li el fet de que com és una funcionalitat del llenguatge que s'ha estandarditzat recentment, la majoria de les implementacions són inconsistents entre elles i moltes tenen greus errors en el seu tractament.

Com que, en l'actualitat, cap implementació de C++ disponible implementa la paraula clau `export` que possibilitaria separar les declaracions i les implementacions dels símbols templatitzats, els temps de compilació d'una aplicació que faci servir la llibreria són molt llargs i els executables es fan massa grans.

6.3.4 Comprovació de conceptes

Fent servir templates en C++ per programar de forma genèrica, el compilador comprova, quan els fas servir, si els conceptes estan alineats amb els objectes que s'han decidit en temps de compilació.

Amb els templates, la comprovació de concepte es fa en temps de compilació, de fet, però els errors que normalment donen els compiladors són molt críptics en el cas de no estar alineats.

Alguns mecanismes de metaprogramació permeten convertir aquests errors en errors identificables.

McNamara [63] introdueix alguns mecanismes de comprovació de conceptes amb templates i en recull d'altres, que es fan servir, per exemple, a la llibreria Boost[3], per fer-ne una avaluació dels avantatges i dels inconvenients.

Una d'aquestes aproximacions a la comprovació de conceptes va ser proposada per Strouptrup[75] i es pot esquematitzar en el llistat 6.1.

El que té de bo aquesta tècnica és que l'error es dóna just en el punt d'ús del template i és una tècnica oberta[71]. McNamara anomena a aquest tipus de comprovació de concepte, comprovació estructural, donat que el que s'està comprovant és que es puguin fer un seguit d'operacions. Les instanciacions dels templates que realment fan servir les expressions són mitjançant referències no pas per crida. Amb això s'aconsegueix que els compiladors l'instanciin i, d'aquesta manera es comprovi el concepte, però, un cop instanciada, el codi relacionat s'elimini com a codi mort.

McNamara diu que l'aproximació d'Strostrup va molt bé per permetre que classes de terceres parts sigui incloses com part del concepte, però, que aquesta relaxació de les restriccions fa que puguin donar-se falsos positius, per exemple amb conversions implícites que poden falsejar el tipus de retorn de les expressions. McNamara proposa una modificació a la solució d'Stroustrup basada en uns tipus templates els constructors dels quals només accepten el tipus desitjat (el template) com a paràmetre (en l'exemple un `bool`). Segons l'estàndard, en aquest cas no pot haver més conversions

```
template <typename T, typename Concept>
inline void Require() {
   (void) ((void (Concept::*)(T)) &Concept::constraints<T>);
}
template <typename T>
struct LessThanComparable {
   T y;
   template <class Self>
   void constraints(Self x) {
      bool b = x<y;
      (void) b;        // suppress spurious warning
   }
};
template <typename T>
void Sort ( T & container) {
   // just inside any template that wants to ensure the requirement:
   Require< T::element, LessThanComparable<T::element> >();
   ...
}
```

Llistat 6.1: Comprovació de conceptes segons Stroustrup

implícites.

McNamara proposa, també, una aproximació (llistat 6.3) totalment oposada a la de Stroustrup. Proposa interfícies estàtiques, quelcom semblant a les classes abstractes però, que podem fer servir quan no necessitem polimorfisme dinàmic, i l'únic que volem es polimorfisme estàtic, és a dir, quan volem treballar amb conceptes per fer servir templates. A aquesta aproximació, McNamara l'anomena comprovació nominal, donat que el que es comprova es que la declaració de la classe.

Donat que troba que cap dels dos models de comprovació de concepte es infal·lible, McNamara barreja en una implementació final els dos tipus de comprovació de conceptes, en la que, donat un concepte podem escollir quin tipus de restricció de concepte fem servir.

Un bon avantatge de la implementació de McNamara, i que faig servir intensivament a la implementació del suport XML, és que, amb el mateix procediment amb el que genera l'error es podria escollir en temps de compilació una implementació alternativa

```
template <typename T, typename Concept>
inline void Require() {
   (void) ((void (Concept::*)(T)) &Concept::constraints<T>);
}
template <typename T>
struct LessThanComparable {
   T y;
   template <class Self>
   void constraints(Self x) {
      bool b = x<y;
      (void) b;        // suppress spurious warning
   }
};
template <typename T>
void Sort ( T & container) {
   // just inside any template that wants to ensure the requirement:
   Require< T::element, LessThanComparable<T::element> >();
   ...
}
```

Llistat 6.2: Millora de la comprovació de conceptes d'Stroustrup

segons els conceptes que el tipus implementi. A aquesta característica se li anomena *static dispatch* o selecció estàtica del mètode.

L'inconvenient que trobo en aquesta aproximació és que cal especificar la pertanyença a tipus en la mateixa declaració de la classe. Això fa que no sigui usable amb tipus nadius del C o amb classes dissenyades per tercers, a més d'embrutar la definició de la classe fent-la poc llegible per als usuaris que ho veuen per primer cop.

A CLAM he proposat una solució semblant a la nominal, però, externa al tipus instanciant per a un tipus donat la classe template que representa el concepte.

La solució de McNamara fa servir algunes tècniques que es fan servir per metaprogramar amb templates. Per exemple, la tècnica que es fa servir per fer apareixer errors de metaprograma amb sentit en temps de compilació. Es basa en fer servir un membre d'un template que no existeix, i on el membre té com a nom el text de l'error.

```
template <class T> struct LessThanComparable {
   template <class Self>
   struct Traits {
      static const bool valid = false;
   };
   template<class Self> static void check_structural() {
      bool (Self::*x)(const T&) const = &Self::operator<;
      (void) x;  // suppress "unused variable" warning
   }
protected:
   ~LessThanComparable() {}
};

struct Foo : public LessThanComparable<Foo> {
   bool operator<( const Foo& ) const { ... }
   // whatever other stuff
};

template <class T> inline void Sort( vector<T>& v ) {
   SortHelper< StaticIsA<T,LessThanComparable<T> >::valid >::doIt( v );
}

template <bool b> struct SortHelper;

template <> struct SortHelper<true> {
   template <class T> static inline void doIt( vector<T>& v ) {
      // actually sort the vector, using operator< as the comparator
   }
};
template <class T>
struct Error {};

template <> struct SortHelper<false> {
   template <class T> static inline void doIt( vector<T>& ) {
      Error<T>::Sort_only_works_on_LessThanComparables;
   }
};
```

Llistat 6.3: Static interfaces d'en McNamara

```
ConceptCheck<T>::Error_Has_Fet_Servir_Un_Objecte_No_Ordenable;
```

En tots els compiladors (aquest cop sí) dóna un error que indica que:

```
Error_Has_Fet_Servir_Un_Objecte_No_Ordenable is not a member
of ConceptCheck<YourUnordenableClass>
```

És un error inherent al codi que hi ha escrit, però, afortunadament, aquesta vegada és útil el fet que el compilador mai instanciï el codi fins que no es fa servir.

6.3.5 Standard Template Library

La programació genèrica ha pres importància a l'escena de la programació en C++ arran de la introducció a especificació de l'estàndard de C++ de la STL (*Standard Template Library*). La STL és una llibreria que fa servir programació genèrica amb templates per oferir un seguit d'abstraccions i mecanismes que faciliten la codificació.

A més de fer servir el codi de STL, en molts casos CLAM ha adoptat la seva forma de fer les coses, donat que les diferents implementacións fan servir algunes solucions molt intel·ligents a alguns problemes que nosaltres mateixos ens hem trobat.

Els elements més utilitzats de la STL són els contenidors genèrics. Un contenidor és una estructura de dades que emmagatzema elements del mateix tipus, per exemple, una llista, un vector, un arbre, una pila...

Aquestes estructures tenen diferent organització interna. Per poder abstraure algorismes que puguin aplicar-se als contenidors d'una forma genèrica, es fa servir entre d'altres coses el concepte d'iterador.

Un iterador és quelcom que ens serveix per accedir als diferents elements d'un contenidor. Els contenidors han de proveir mètodes per obtenir iteradors i els algorismes ha de saber manegar-los. D'aquesta manera, es poden desacoblar les dependències d'implementació entre els algorismes i els contenidors.

Iteradors

Els iteradors són objectes que es defineixen per a cada contenidor concret però que ofereixen una mateixa interfície estàtica. A més, poden encapsular cadascun diversos tipus de recorregut sobre una mateixa estructura.

La STL modela els iteradors imitant la interfície que es fa servir per accedir a una estructura d'array amb punters de C, sent els punters els iteradors de l'estructura.

Amb una estructura array de C faríem:

```
void strncpy(char * origen, int n, char * desti)
{
  char *finalOrigen=origen+N;
  while (origen!=finalOrigen)
    *desti++ = *origen++;
}
```

Per uniformitat i per economia de conceptes, les operacions que es fan servir

- Els iteradors s'incrementen i decrementen amb els operadors `++` i `--` de forma prefixa i posfixa per avançar o retrocedir un element.
- També els podem sumar o restar un sencer N per avançar o retrocedir N elements.
- Per accedir al valor actual fem servir l'operador unari `*`, si volem accedir a un camp del valor, també podem fer servir l'operador `->`, tot igual que si fos un apuntador.
- Amb els operadors `==` i `!=`, podem comparar dos iteradors per veure si apunten al mateix element o no.

Un iterador és un objecte que implementa aquestes funcions. De tal forma que si implementem la funció:

```
template <class iterator>
void copy_n(iterator origen, int N, iterator desti) {
  iterator finalOrigen = origen + N;
  while (origen != finalOrigen)
    *desti++ = *origen++;
}
```

Alguns iteradors, però, són més restrictius envers les operacions que poden realitzar. Segons aquestes restriccions es defineixen diverses categories d'iteradors. La funció iterator `category(iterador)` retorna un punter nul del tipus de tag que correspon amb la seva categoria.

struct output iterator tag; Només es pot fer una sola inserció de valor entre increment i increment. No es pot llegir el valor apuntat, només escriure-hi.

struct input iterator tag; Es pot extreure una posici´o tantes vegades com vulguem mentre no retorni l'element singular que indica que no podem extreure'n més. Es pot incrementar i llegir d'una altra nova posició però, una vegada incrementat les altres còpies de l'iterador es tornen inconsistents. No es pot escriure a la posició apuntada, només llegir-hi.

struct forward iterator tag : public struct input iterator tag; Es pot fer servir on es fa servir un output iterator escrivint o on es fa servir un input iterator llegint. Si escrius i després llegeixes, obtens el valor que has escrit. Les copies d'un fordward iterator funcionen de forma consistent independentment.

struct bidirectional iterator tag : public struct forward iterator tag; Es pot fer servir on es fa servir un forward iterator i, a més, permet decrements.

struct random access iterator tag : public struct bidirectional iterator tag; Es pot fer servir on es fa servir un *bidirectional iterator* i, a més, a pot fer aritmètica sencera de la mateixa manera que fem amb els punters. Podem fer servir l'operador d'indexació o l'operador de suma i resta amb sencers.

Aquests tags es poden fer servir per determinar en temps de compilacions com implementar alguns algorismes. Si sobrecarreguem funcions per un paràmetre que sigui un punter al tipus de tag podem escollir la implementació a partir del aparentment inútil valor de retorn de la funció `iterator_category`. És una altra forma de fer un *stàtic dispatch*

```
template <class iterator>
iterator advance(iterator origen, int N)
{
    // Crida a la sobrecarregada segons un tercer par ametre
    advance(origen, N, iterator_category(origen));
}
template <class iterator>
iterator advance(iterator origen, int N, random_iterator_tag * foo)
{
    // Com que es un random iterator podem fer la suma
    return origen + N;
}
template <class iterator>
iterator advance(iterator origen, int N, bidirectional_iterator_tag * foo)
{
    // Com que no es un random ho hem de fer seq¨uencialment
    if (N>=0)
        while (N--) origen++;
    else
        while (N--) origen--;
}
```

Llistat 6.4: Fent servir la categoria de l'iterador

Altres possibilitats dels iteradors que van més enllà del que proporciona la STL, són les que en deriven de construir iteradors alternatius amb diverses finalitats. Baus i Berek [27] proposen alguns iteradors molt interessants que permeten accedir no exactament als elements d'un contenidor:

Iteradors amb funcions: Són iteradors que no retornen pas un element del contenidor sinó el valor retornat per una funció, que pot ser pròpia de l'iterador, o explicitada amb un functor com a paràmetre template.

Iteradors de combinació: Un iterador de combinació és un que encapsula N it-

eradors i una funció de combinació. Les operacions es propaguen als iteradors encapsulats i en demanar un contingut, es retorna el resultat de la funció aplicat als elements apuntats per cadascun dels iteradors.

Iteradors de derreferència: Sovint es construeixen contenidors amb tipus polimòrfics. Aquests contenidors necessiten que l'element bàsic sigui un punter. Molts dels algorismes genèrics operen amb funcions que es poden aplicar al tipus dels objectes iterats, però, quan el contenidor conté punters, el tipus dels objectes iterats és un punter i les funcions a aplicar són pròpies del tipus al que apunta el punter.

Iteradors sobre membres: Els iteradors sobre membres permeten accedir directament als membres mitjançant un iterador.

Iteradors generadors: Els iteradors generadors no s'extreuen d'un contenidor sinó que permet iterar sobre una seqüència de nombres generada. Són molt útils per aprofitar algorismes genèrics, per exemple, per aplicar una funció binària entre un vector de enters i una sèrie de nombres calculada (fibonacci...).

Altres implementacions del concepte d'iterador permeten fer recorreguts diferents d'un mateix contenidor. Stroustrup [74] defineix formes d'accedir a Slices. Un slice és un DECORATOR que aplicat a un vector numèric que representi una matriu, permet simular recorreguts per les seves files i les seves columnes amb molt poc cost. Siek defineix a la seva llibreria MTL [70] diferents iteradors per accedir amb diferents recorreguts a matrius multidimensionals amb representacions molt diverses.

Objectes funcionals: *Functors*

Les classes que implementen l'operador de crida (operator()) ofereixen la mateixa funcionalitat que els punters a funció, peró, a més, poden contenir dades que es facin servir dins de la crida. A aquestes estructures es diuen objectes funcionals o functors. Posem com a exemple el listat 6.5. Visualitzaria un 8 i un 15 per la sortida estàndard. És com si tinguèssim una funció configurable.

```
#include <iostream>
class Sumador {
    int sumand;
public:
    Sumador(int unEnter=0) {
        sumand = unEnter;
    }
    sumand(int unEnter=0) {
        sumand = unEnter;
    }
    int operator() (int & operand) {
        return operand+=sumand;
    }

};
int main()
{
    Sumador funcioIncrementadora(3);
    cout << funcioIncrementadora(5) << endl;
    funcioIncrementadora.sumand(10);
    cout << funcioIncrementadora(5) << endl;
}
```

Llistat 6.5: Un functor

```
void aplica(int * inici, unsigned int N, void (*f) (int &))
{
    while (N--)
        f(*(inici++));
}
```

Llistat 6.6: Funció 'aplica' amb punters a funció

La funció aplica definida al llistat 6.6 permet aplicar una funció f a tots els elements d'un array d'enters. Fent servir templates per al tipus de f (llistat 6.7), es pot generalitzar la funció aplica pel concepte d'un objecte que defineixi l'operador de crida amb els paràmetres adients, com per exemple, el functor `Sumador`.

```
template<class iterator, class functor>
void aplica(iterator inici, itertor fi, functor f)
{
    while (inici!=fi)
        f(*inici++);
}
```

Llistat 6.7: Funció 'aplica' genèrica

Molts algorismes de la llibreria estàndard fan servir aquesta generalització per tenir més flexibilitat. La STL defineix alguns functors especials que permeten fer combinacións d'objectes.

El concepte de functor a la STL també requereix altres coses com per exemple que defineixi tipus interns per indicar el tipus de retorn o els paràmetres que accepta. D'aquesta manera es facilita l'ús des dels algorismes, que tot i que el functor sigui template poden obtenir aquesta informació, suposant que es compleixi el concepte.

6.4 Programació generativa

He intentat separar, en diferents apartats d'aquest capítol, tècniques de la programació genèrica i tècniques de la programació generativa. Les dos estan molt lligades per les eines que fan servir en la seva implementació C++: els templates. També estan lligades pel fet de que moltes de les mancances del C++ per a la programació genèrica es fan mitjançant els mateixos mecanismes que es fan servir per metaprogramar amb templates i que al metaprogramar es fan servir conceptes genèrics. És per això que la literatura que he trobat al respecte anomenava de forma diferents a coses molt semblants.

En cap cas, ha estat la meva intenció que la classificació que he fet entre tècniques genèriques i tècniques generatives fos una classificació categòrica. Molts d'aquests conceptes són emergents i en encara no semblen prou establerts per haver-hi una nomenclatura consolidada.

6.4.1 Metaprogramació

Un metaprograma és un programa que genera un altre programa. És a dir, un metaprograma manipula i genera programes com si aquests fossin les dades. Generalment, l'objectiu de metaprogramar és obtenir un programa optimitzat per a unes certes condicions d'execució. És a dir, el metaprograma s'executa la primera vegada per generar el programa i a partir d'aquí és el programa optimitzat el que s'executa.

Aquest concepte prové de llenguatges com Lisp o Smalltalk i els seus descendents. Per aquests llenguatges els programes es poden tractar com a dades de forma natural. En Lisp, són llistes de símbols, i, en Smalltalk, són objectes manipulables.

Amb metaprogramació es pot aconseguir en temps de compilació el mateix que es pot arribar a aconseguir, en un framework orientat a components, en temps d'instal·lació amb el procés de configuració i parametrització dels components[32].

Les diferències entre la parametrització dinàmica (configuració) i la parametrització estàtica (metaprogramació), és que en la darrera el programa final no ha de considerar les diferències de configuració i pot estar optimitzat per a una configuració concreta.

Per exemple, un algorisme necessita conèixer el factorial de N on N és un dels paràmetres de configuració. Si, per a una configuració concreta sabem que N val 8, llavors es podria substituir `factorial(N)` pel seu valor final i s'optimitzaria. Però, fent aquesta substitució directament al codi, perdem generalitat i, quan N no valgui 8, caldrà canviar el codi font. Podem, però, deixar que sigui el metaprograma qui calculi aquest factorial i que el programa com a tal el faci servir com a constant.

Com a contrapunt, la metaprogramació no té la flexibilitat de canviar en temps d'execució, però, això no vol dir pas que no es puguin combinar ambdues tècniques

per poder-se adaptar als requeriments.

6.4.2 Metaprogramació i el disseny de llibreries

La metaprogramació és un recurs que ataca directament un dels punts més febles del disseny de llibreries.

Hom, quan dissenya una llibreria, no sap quin serà l'entorn on es farà servir cadascuna de les seves funcionalitats. En conseqüència, es desconeix quins dels casos cal optimitzar i es tendeix a escollir la solució menys dolenta per tots els casos o la que s'ajusta al cas més freqüent.

La metaprogramació permet que els algorismes implementats s'ajustin a cada cas quan l'usuari de la llibreria els fa servir.

La metaprogramació també és útil per generar *codi d'usuari*. S'entén per codi d'usuari, per codi que no hauria de ser part de la llibreria, sinó part de l'ús concret que en fa l'usuari. Per exemple, si l'usuari deriva una classe d'una que en prové de la llibreria, hi ha certes coses que cal fer a la classe. Aquest seguit de coses potser són massa complexes per a ús directe de l'usuari.

En aquests casos la llibreria pot proporcionar una combinació de macros de preprocessat i metaprogramació per generar el codi.

6.4.3 Metaprogramació amb templates

En entorns com Lisp o Smalltalk, el codi que genera la metaprogramació és viu i es pot executar de forma directa dintre del mateix fil d'execució que el metaprograma.

En llenguatges com C++, en que no es pot afegir codi a un programa en temps d'execució[1], l'execució del metaprograma cal fer-la abans de l'execució del programa

[1]Evidentment estem obviant la càrrega dinàmica de llibreries que no és una caracterìstica pròpia del llenguatge.

final. Això es pot fer, per exemple, amb un programa que generi codi font en C++, però, C++ ofereix mecanismes que permeten fer servir el compilador com intèrpret del metaprograma.

Algunes tècniques permeten fer servir també el preprocessador de C per metaprogramar[4], però, ens centrarem aquí en algunes tècniques de metaprogramació basades en templates.

Els templates es varen afegir al llenguatge de programació C++ amb la intenció de permetre la programació genèrica de contenidors i algorismes (Vegeu l'apartat 6.3). Malgrat aquest objectiu, algunes característiques dels templates i els mecanismes amb els que s'implementen, han permès desenvolupar tècniques de metaprogramació.

Les tècniques de metaprogramació conformen un meta-llenguatge que permet construir amb la sintaxi del C++, estructures típiques dels llenguatges funcionals com Scheme.

Podem fer servir classes o estructures template per contenir les funcions i les especialitzacions d'aquestes estructures templatitzades, per programar alternatives i condicionals de les funcions segons els paràmetres del template.

La instanciació del template adopta el rol de crida a la funció. L'execució del codi es pot fer mitjançant els membres de les classes:

- Constants estàtiques enteres
- Enumeracions
- Funcions estàtiques *inline*
- Declaracions `typedef`
- Definicions de classes internes

Les constants estàtiques no funcionen gaire bé en VisualC, però, les enumeracions són les perfectes substitutes. Ambdues poden, segons l'estàndard definir dintre de la

mateixa classe el seu valor. Aquest valor es pot calcular de forma directa amb una expressió literal, o bé fent servir un membre de una altre classe template que caldria instanciar.

El mateix passa amb les funcions inline, tot i que les funcions inline depenen més de l'optimització que faci el compilador.

Tant els valors enters com els tipus poden servir de paràmetre per a un template, per això, les classes internes i les definicions d'alies de tipus són tan importants en metaprogramació. Serveixen per instanciar noves estructures templates.

6.4.4 Càlculs en temps de compilació

La metaprogramació amb templates fa servir el procés de resolució de templates del compilador per interpretar el metaprograma. La idea es veu molt clarament en el llistat 6.8 que és un exemple de com es poden calcular valors en temps de compilació[78]. En aquest cas, es calcula un factorial. Aquest càlcul es fa tot encadenant les instanciacions dels templates. La instanciació del template `factorial<20>` força l'instanciació del template `factorial<19>` i així, successivament fins que arriba finalment a la instanciació explícita per a `N=0`.

Els paràmetres templates estan limitats a nombres enters, i els valors de retorn en forma d'`enums` tenen el mateix problema. Podem fer servir els paràmetres enters per representar no nombres reals però sí fraccions. També podem fer servir funcions inline que retornin `double` i es puguin calcular a partir de constants.

Un exemple un xic més complicat, que il·lustra aquesta tècnica és el llistat 6.9 que calcula funcions trigonomètriques mitjançant sèries de Taylor. És molt més òptim que fer taules de cerca si es coneixen en temps de compilació els angles a calcular.

```
#include <iostream>

// El cas general
template <int N>
struct factorial {
    enum { value=N*factorial<N-1>::value };
};

// Instanciem explícitament el cas singular
template <>
struct factorial<0> {
    enum { value=1};
};

// El factorial es calcula en compilació
// És un immediat en execució (O(1))
int main () {
    std::cout << " El factorial de 20 és"
          << factorial<20>::value << std::endl;
    return 0;
}
```

Llistat 6.8: Metafactorial

```
#include <iostream>
#include <cmath>

// Calcula el terme K de J termes de la sèrie de Taylor.
template<int N, int I, int J, int K>
class TaylorSerie {
public:
    static double sin() {
        return 1-(I*2*M_PI/N)*(I*2*M_PI/N)/(2*(J-K)+2)/(2*(J-K)+3) *
            TaylorSerie<N,I,J,(K-1)>::sin();
    }
};

// Especialització pel cas trivial del bucle
template <int N, int I, int J>
class TaylorSerie<N,I,J,0> {
public:
    static double sin() {
        return 1;
    }
};

// sin(x) on x=2 * M_PI * I/N
// El darrer paràmetre és la precisió mesurada amb el nombre de termes
template<int N, int I, int J = 40>
class Trigon {
public:
    static double sin() {
        return (I*2*M_PI/N) * TaylorSerie <N,I,J,J>::sin();
    }
    // Nota: Aqui podriem incloure les funcions cos, tg...
};

void main () {
    // Aquesta crida esdevé un seguit de crides recursives
    // que es fan inline i, com l'expressió resultant és
    // constant, es calcula en temps de compilació
    std::cout << Trigon<360,43>::sin() << std::endl;
}
```

Llistat 6.9: Metatrigon

6.4.5 Operacions simbòliques amb tipus

A més, de fer càlculs en temps de compilació. també es fan servir els tipus per càlcul simbòlic. Un exemple simple de com ver servir els tipus d'aquesta manera es va veure a l'apartat 6.3.5, quan s'explicava com conèixer en temps de compilació les operacions que suportava l'iterador que es feia servir en un algorisme genèric pel tipus d'iterador i fer servir un mètode o un altre.

Aquest tipus de resolució pot arribar a ser més complexa. Com per exemple, la resolució simbòlica de derivades [49].

6.4.6 Meta-expressions template

Un altra tècnica de metaprogramació molt estesa pels beneficis d'eficiència que comporta, són les *template metaexpressions*. Les meta-expressions permeten l'avaluació tardana d'expressions tot considerant les expressions com a tipus i permetre la seva combinació per construir expressions-tipus més complexes.

Això es fa tot combinant polimorfisme estàtic i dinàmic que permeten la execució estàtica o dinàmica quan cal.

Les metaexpressions també possibiliten la seva utilització com si fossin lambdes de Lisp o functors d'STL.

L'avaluació tardana d'expressions (*Lacy evaluation*) permet optimitzar molt l'execució d'alguns algorismes amb estructures grans.

Per exemple, a l'aproximació orientada a objectes de l'expressió `R=A+B+C`, on les variables són vectors, es defineix la operació suma pels vectors de tal manera que de la suma entre A i B es genera un quart objecte vector que és sumat a C per generar el resultat. A més, si guardem el resultat en R, això implica una còpia de l'objecte.

Si ho programèssim amb un bucle accedint a cada valor ho faríem simplement posant a `R[i]` el resultat de `A[i]+B[i]+C[i]`. Amb metaexpresions, les operacions no donen

com a resultat un valor sinó una expressió que no es calcula fins que es fa l'assignació i es fa de la manera òptima.

En resum, amb les template metaexpressions mantenim l'expressivitat de l'orientació a objectes però, amb l'eficiència de construir a mà el bucle.

Es pot pensar que aquesta aproximació es molt limitada, però, la seva potència és molt gran. Hi ha tres exemples molt il·lustratius del que es pot arribar a fer amb metaexpressions.

- **Blitz++[2]:** És una llibreria d'algebra lineal d'alt rendiment que fa servir metaexpressions i metaprogramació per optimitzar els càlculs. El codi font de Blitz++ està ple de solucions enginyoses per resoldre casos de metaprogramació.
- **SEMT[49]:** Fa servir template metaexpressions per fer derivació simbòlica d'expressions algebràiques.
- **Spirit[13]:** És una sorprenent llibreria per programar compiladors on les produccions són metaexpressions en C++.

6.4.7 Inconvenients

Fer servir el llenguatge d'una forma tan enrevessada té alguns inconvenients:

- La sintaxi de C++ no està pensada per aquest tipus de programació i sovint els metaprogrames esdevenen complexos de llegir sobretot per algú que no estigui acostumat.
- Els compiladors tampoc estan pensats per fer-se servir com a interprets de metaprogrames. Sovint un algorisme metaprogramat $O(n)$ es pot convertir en temps de compilació en un algorisme $O(n^3)$ degut, per exemple, a que executant operacions unitàries el compilador crida l'algorisme de cerca de templates instanciats. A vegades cal moderar els càlculs que es metaprogramen, per mantenir un temps de compilació raonable.

- Igual que amb la programació genèrica, hi ha molts compiladors com el Visual que no suporten templates gaire bé i disten molt d'acomplir els mínims de l'estàndard.

6.5 Resum

En aquest capítol s'han explicat diverses tècniques i metodologies que s'han fet servir per dissenyar i implementar el projecte.

L'orientació a objectes ha servit com a base del disseny de projecte. En aquest sentit els patrons de disseny són una eina molt útil per trobar solucions a problemes de disseny i avaluar la seva bondat.

També he introduït alguns conceptes que es surten un xic de l'orientació a objectes:

La programació genèrica ofereix una visió més amplia de la reutilització del codi tot reduint les restriccions per fer-ho. La programació generativa ofereix un nou nivell de flexibilitat, en temps de compilació, a l'hora de fixar les decisions de disseny.

Concretament, al projecte, s'han definit alguns conceptes (en el sentit de la programació generativa) per poder fer static dispatch segons quins conceptes tinguem.

També s'ha fet servir intensivament la STL, la llibreria genèrica que forma part de la llibreria estàndard. Alguns conceptes de la STL com ara iterador, functor o container no només els he fet servir, sinó que els he incorporat com a part del meu disseny.

Les tècniques de metaprogramació, s'han fet servir, per facilitar la tasca de l'usuari a l'hora de fer servir el suport XML, tot generant codi òptim personalitzat a les seves necessitats com s'explica a l'apartat 7.4.

Capítol 7

Realització del projecte

7.1 Metodologia i entorn de treball

7.1.1 Recursos

Equip humà

Durant el transcurs d'aquest projecte, el equip de desenvolupament de CLAM ha estat format per 6 persones. Cadascun dels membres de l'equip treballa amb una o dues plataformes i està moderadament especialitzat en certes àrees de CLAM. Aquesta especialització és moderada perquè no és exclusiva. De fet, el disseny i el desenvolupament del sistema en sí és molt creuat entre els diferents desenvolupadors.

Els sis desenvolupadors del projecte són:

- Xavier Amatriain (Cap de projecte)
- Pau Arumí (Dynamic Types)
- Enrique Robledo (Encapsulat Processos)
- Miquel Ramírez (Interfícies)

- Maarten Deboer (Subsistemes d'àudio)
- David García (XML)

Entre parèntesis s'indica la seva especialitat tot i que com ja s'ha comentat aquestes especialitats són molt difuses: hi ha molts àmbits que no queden cobertes per aquestes àrees estan coberts per tothom i tothom ha intervingut en el disseny i implementació de les àrees dels altres.

Darrerament s'han incorporat tres desenvolupadors més (Albert Mora, Xavier Rubio i Sandra Gilabert) que actualment estan en fase de formació però, que ja comencen a donar resultats.

També cal mencionar una desena d'usuaris que treballen en altres projectes del grup d'investigació. Aquests usuaris col·laboren incorporant a CLAM els nous algorismes desenvolupats en l'àmbit dels seus projecte i, a més, fan de beta-testers de les successives entregues de CLAM.

Servidors

L'equip de desenvolupament fa servir dos hosts Debian GNU/Linux com a servidors. Un conté serveis per al grup d'investigació en general i fa les següents funcions:

- Servidor de serveis de xarxa (NIS)
- Servidor de fitxers remots (NFS i Samba)
- Servidor de correu
- Groupware (BSCW)

L'altre servidor, està més orientat a tasques pròpies de desenvolupament:

- Repositori de codi (Servidor de CVS)

- BugTracking (Mantis)
- Accés CVS de només consulta (ViewCVS)
- Generador de documentació de referència (Doxygen)

Ambdós servidors estan assegurats amb un sistema de còpies de resguard.

Estacions de treball

Les d'estacions de treball són PC's. Per tenir representades en el grup de treball el major nombre possible de plataformes suportades, les estacions de treball funcionaven amb diversitat de sistemes MS-Windows i GNU/Linux.

La major part dels sistemes MS-Windows amb els que es treballaven són *Windows 2000* tot i que també es manté alguna estació NT4 i la major part dels sistemes GNU/Linux són distribucions Debian, tot i que també cal mantenir alguna estació amb RedHat per comprovar que CLAM també hi funcionava i per generar paquets d'aquesta distribució.

També es disposa d'un G4 amb MacOSX per fer proves i compilacions en aquesta plataforma. Tot i així, la portabilitat sobre Mac s'ha mantingut sobretot a nivell de compilador, donat que el compilador CodeWarrior està disponible també per a Windows i Linux. En la estació G4 s'han fet molt poques proves, relacionades, sobretot, amb els mòduls d'àudio i MIDI, que encara no funcionen correctament.

Programari

Els compiladors suportats són VisualC versió 6 a Windows i GCC 2.95 a Linux. També s'han provat sense problemes, tot i que no es manté el software en aquests compiladors:

- VisualC++ versió 7 per Windows

- GCC 3.0 a Linux
- CodeWarrior versiói 7.0 per Windows

Cal dir que els compiladors són el punt més dur del disseny d'aquesta llibreria, donat que la majoria no implementen correctament l'estàndard.

El cas més alarmant és el VisualC. A la pagina de la llibreria Boost [12] hi ha un recull de bugs del Visual fan que compilar porcions codi estàndard com les que apareixen al llistat 7.1 sigui impossible.

```
// Wrong explicit function template instantiation

template<class T>
void f()
{
  printf("%d\n", sizeof(T));
}

void use()
{
  f<double>();      // output: "1"
  f<char>();        // output: "1"
  return 0;
}


// Partial specialization of class templates does not work.

template<class T>
struct A {};

template<class T>
struct B {};

template<class T>
struct A<B<T> > {};  // template class was already defined as a non-template
```

Llistat 7.1: Codi estàndard que no compila en Visual

Vam provar la versió 7 per veure si funcionava millor però, la nova versió, en comptes de quedar-se penjat als casos anteriors, donaven errors que no tenien cap sentit.

La llibreria 'estàndard' que ofereix el Visual està basada en una versió antiga de la llibreria de HP que té moltes falles. Per compilar sense problemes, els desenvolupadors de Windows fan servir la STLPort, que és una implementació de la llibreria que està pensada per suplir la pròpia del compilador en aquests casos.

Comparat amb el VisualC, el GCC 2.95 és molt més complidor amb l'estàndard, però encara no ho és completament. Les versions 3.0 són més fidels però, no les fem servir normalment per desenvolupar perquè ha canviat l'ABI i el depurador GDB encara no està preparat per aquest canvi.

7.1.2 Gestió de desenvolupament concurrent

Quan diversos desenvolupadors estan actualitzant un mateix codi, sorgeix el problema de gestionar les inconsistències entre les diferents modificacions. En aquest sentit, la comunicació entre els desenvolupadors és clau, però, es pot fer més àgil dotant a l'equip d'algunes eines.

CVS

L'eina més important és un sistema de gestió de versions concurrents. Pel desenvolupament de CLAM s'ha fet servir CVS. CVS és un sistema de gestió de versions molt simple, però, al mateix molt versàtil i útil. Avui dia es fa servir a la majoria de projectes on els desenvolupadors estan distribuïts geogràficament.

La funció principal d'aquest programari és evitar els conflictes entre les modificacions dels diversos programadors. Cada desenvolupador té una còpia del codi en local on fa les seves modificacions i que pot sincronitzar amb l'estat del repositori central en qualsevol moment. Quan el desenvolupador està satisfet amb les seves modificacions, les confirma al servidor. Aquest no li deixarà fer-ho si les modificacions entren en conflicte amb altres que el desenvolupador encara no ha sincronitzat.

Al repositori central, queda, a més, un històric documentat dels canvis de manera que

es permet revisar tots els canvis que s'han fet en cada arxiu, o desfer canvis concrets.

Una altra funcionalitat addicional de CVS és la capacitat de crear línies (branques) de desenvolupament independents sobre un mateix codi.

Reunions

CVS només soluciona la concurrència a nivell de codi font. També cal que els desenvolupadors es sincronitzin a nivell de disseny. Per això, cal un altre tipus de sincronització que es reforça amb reunions formals i informals.

Les reunions formals es fan amb periodicitat setmanal amb l'objectiu de gestionar els esforços de desenvolupament. Les reunions informals es produeixen ocasionalment quan sorgeix la necessitat de dissenyar quelcom de forma conjunta, aclarir puntualment algun punt o consensuar alguna decisió de disseny presa unilateralment abans de tirar-la endavant.

Discussió online

El problema de les reunions informals és que:

- interrompien el desenvolupament que cadascú portava a terme
- sorgien massa freqüentment, i,
- a les actes no quedava constància de tot el que s'hi discutia

Per aquests motius, tot i estar tots els desenvolupadors treballant a la mateixa sala, es va considerar raonable establir mecanismes per la discussió en línia mitjançant els fòrums del software de groupware (BSCW[5]) que es fa servir a l'MTG i diverses llistes de correu temàtiques que es sincronitzaven amb els fòrums.

Aquestes discussions en línia varen suavitzar la freqüència de les interrupcions i en deixaven constància tant per a consulta posterior com per als que no n'havien pres

part.

7.1.3 Gestió de tasques pendents i correcció d'errors

A mesura que la llibreria ha esdevingut més complexa, i, ha crescut el nombre d'usuaris, ha calgut adoptar un sistema de gestió d'informes d'error i de propostes de funcionalitat.

Hi ha disponibles diverses eines de gestió d'informes d'error. Per a CLAM s'ha escollit Mantis, donat que la gestió del cicle de vida dels informes d'error és molt clara. Es pot establir una discussió sobre aquests bugs, i té una interfície pensada per avaluar el cost del canvi i l'assignació de recursos, tot plegat informant via correu electrònic als desenvolupadors interessats.

A més, Mantis és un programari lliure, net, madur i molt fàcil d'instal·lar i administrar, al contrari d'altres productes no comercials semblants que vam trobar.

Les alternatives a Mantis que vàrem provar van ser Bugzilla i phpBT.

Bugzilla és molt madur. Es va desenvolupar per recollir els errors en el desenvolupament del navegador web Mozilla. Tot i la seva maduresa, es nota molt que ha estat desenvolupat a pegots. El procés d'instal·lació i configuració és molt complex i fosc.

L'altre programari, phpBT, és molt semblant a Mantis, però, la gestió de correu s'adapta menys a les necessitats que tenim al grup de desenvolupament. A més, ens semblava molt més elegant i flexible la seva interfície i més dinàmic i coherent el seu ritme d'entregues de noves versions. Tot i ser un projecte més jove tenia molt més bona base.

En Mantis els informes d'errors o peticions de funcionalitat, adopten els següents estats al llarg de la seva vida:

new: L'informe d'error ha entrat en el sistema

feedback: L'informe no es prou entenedor, i requereix mes informació per part del

informador.

acknoledged: Algun desenvolupador l'ha vist però, no ha pres cap acció concreta.

confirmed: Un desenvolupador ha reproduit i confirmat l'error informat

assigned: Un desenvolupador ha asumit la tasca de corregir-ho.

resolved: Les accions correctives corresponents a l'error ja s'han pres.

closed: L'informador (o algú altre diferent de qui ha fet el cambi) ha confirmat la desaparició de l'error.

7.1.4 Documentació

La documentació és quelcom important a qualsevol projecte software. En el cas d'una llibreria la documentació pren una importància crucial, donat que s'hi juga, no només el seu manteniment, sinó la seva usabilitat.

Manuals i tutorials

La documentació de CLAM ha d'anar adreçada a tres actors principals amb diferents nivells de profunditat:

Mantenidors: Són aquells desenvolupadors que més endavant mantindran CLAM. La documentació destinada només a aquests actors, comprèn descripcions detallades del disseny i la implementació i les justificacions de les decisions de disseny presses. També inclou altra documentació com els anàlisis de requeriments, els esborranys de dissenys previs i altra documentació produïda durant el desenvolupament.

Desenvolupadors: Anomenem desenvolupadorsi, tot i que, de fet, els tres actors ho són, a aquells que, sense entrar en les interioritats del disseny de CLAM, implementen nous objectes, de processament o de dades. Aquest nivell de documentació, requereix conèixer els requeriments que aquests objectes tenen.

Usuaris: Són aquells usuaris (de fet són programadors) que faran servir CLAM per construir el seu sistema amb objectes de processament i de dades existents. La documentació per aquest tipus d'usuari es limita a descriure la interfície més externa dels objectes de CLAM, per poder-los fer servir.

S'ha compilat un manual d'usuari que cobreix els dos darrers nivells. La documentació adreçada als mantenidors es manté dintre del software de groupware per a la seva discussió continuada.

Doxygen

A banda dels manuals d'usuari, també hi ha una guia de referència generada amb Doxygen. Doxygen és una eina de documentació de codi semblant a JavaDoc.

La documentació per a Doxygen es manté a dins del codi font amb la qual cosa és més fàcil que el mateix programador la mantingui al dia. A partir d'aquesta documentació ficada al mateix codi i de l'anàlisi del propi codi, Doxygen genera documentació en diversos formats interactius: HTML, PDF, texinfo, man...

Aquesta informació és molt útil per els usuaris de la llibreria donat que els hi dona una idea molt clara de les interfícies que han de fer servir.

Doxygen, però, no només té una funció de documentació sinó que és útil com a eina d'anàlisis per al desenvolupament de CLAM. Només a partir de l'anàlisi del codi, i, sense cap documentació explícita per part del programador, Doxygen ja proporciona:

- Inventari de tots els símbols definits
- Diagrames de herència
- Diagrames d'ús
- Diagrames d'inclusió de capçaleres
- Informació sobre qui fa servir cadascuna de les funcions

- Informació sobre que funcions fan servir cada funció
- Informació sobre qui implementa una determinada funció virtual
- Vista del codi amb referències creuades i reforçament de sintaxi amb colors
- ...

7.1.5 Cicle de desenvolupament

Cicle de vida

Tant CLAM com el projecte s'han desenvolupar amb un cicle de vida en espiral. Amb un objectiu final en ment, a cada etapa s'estableixen els objectius parcials i es fa un cicle de vida semblant al clàssic per assolir aquest objectius.

La metodologia de desenvolupament s'ha vist molt influenciada per les eines de desenvolupament i la metodologia estesa als desenvolupament opensource, la qual es caracteritza per un desenvolupament incremental orientat a entregues. A cada fase, cada desenvolupador planifica els desenvolupaments que entraran en la següent entrega.

El sistema de gestió de canvis ens permet que cada desenvolupador, si ha de fer modificacions que puguin alterar el normal desenvolupament als altres, estableixi una branca de desenvolupament i així integrar els canvis en punts concrets. S'estableixen així petits subprojectes de curta durada amb un o dos desenvolupadors involucrats.

El desenvolupament del projecte de final de carrera ha constat de fases que cal veure com petits subprojectes. La planificació havia de ser subprojecte a subprojecte, donat que l'evolució de CLAM també ha estat pas a pas, i, a més, fins que el projecte no ha anat avançant, no es coneixia la viabilitat de cadascuna de les fases. Les fases de desenvolupament han estat:

- Sistema de composició

- Sistema bàsic de passivació
- Adaptadors XML bàsics i array
- Simbolització d'enums i flags
- Passivació en els dynamic types
- Estructures de dades
- Sistema bàsic d'activació
- Adaptadors
- Chained methods
- Activació als dinàmic types
- Adaptadors de contenidors iterables
- Punt d'entrada: Capceleres XML

Prototipatge i integració

A CLAM s'han fet servir moltes tècniques de programació poc comuns que, tot i estar contemplades per l'estàndard del llenguatge, sovint superen el límit del que la implementació concreta és capaç de fer. Un límit que, sobretot en el cas del VisualStudio està molt per sota del que s'espera d'una implementació comercial tan estesa i amb tants recursos al darrera.

Per fer moltes parts d'aquest projecte, sovint s'ha fet servir una aproximació incremental mitjançant prototipus. Els prototipus anaven afegint característiques que s'anaven provant en les principals plataformes: GCC, CodeWarrior i VisualC++.

Aquestes proves agilitzaven la incorporació de noves característiques en no haver de modificar i compilar tota la llibreria i en localitzar en un codi petit els problemes que un compilador concret podia donar. Tot i la frustració de veure com moltes bones idees no superaven la prova de la portabilitat, vam aconseguir trobar solucions alternatives

per a moltes d'elles. És trist entendre per portabilitat, haver de considerar bugs i divergències gratuïtes de l'estàndard.

Aquesta tasca d'investigació hagués estat impossible de dur-se a terme integrant aquestes proves amb tot el codi de CLAM. Així quan integràvem la solució a dintre de CLAM estàvem gairebé segurs de que funcionaria. Amb prototipus petits es controlen més les interaccions i es compila més ràpid.

Proves i detecció d'errors

Cada classe o cada unitat funcional té el seu propi programa de autoproves. Sense pretendre fer totes les proves exhaustives, la bateria de proves prové del recull de les successives proves que s'han anat fent sobre cada classe o conjunt de classes relacionades.

S'han fet proves unitàries exhaustives de caixa blanca i grisa a les classes més centrals del sistema. Les altres classes menys cèntriques s'ha confiat en les proves de funcionalitat inicials que sovint eren a una prova d'integració. En les classes no centrals, només s'ha passat a fer proves unitàries en el cas que haguem necessitat localitzar un vici concret.

S'ha procedit reactivament a les proves de les classes no cèntriques perquè els requeriments de temps que teniem permitien provar totes les classes de forma exhaustiva.

S'ha seguit la política de, un cop fetes, conservar i mantenir aquestes proves com a una part més del sistema per aprofitar aquest temps invertit en desenvolupar la prova.

També s'ha procurat es que la majoria de les proves, sigui quin sigui el seu origen, puguin interactuar amb un sistema d'auto-proves i que no requereixi la intervenció humana per verificar si han estat superades o no. D'aquesta manera podem passar les proves més sovint i detectar més ràpidament quan una modificació ha introduït inestabilitat.

Altres tres eines han facilitat molt la tasca de detecció de irregularitats:

- Les **comprovacions d'invariant**, que són uns mètodes que l'únic que fan és comprovar que els objectes complexos estiguin en estat intern consistent. Així es poden detectar condicions d'errors latents que trigarien en maniferstar-se o no ho farien pas.
- Les **assertacions** que verifiquen algunes precondicions i postcondicions dels mètodes concrets. El seu funcionament s'explica en un apartat més endavant. Juntament amb les assercions estan els avisos que podem habilitar per detectar situacions, que, tot i no ser incorrectes, si que són sospitoses.
- La **inspecció** que són mètodes que permeten representar les interioritats d'un objecte mentre el programa s'executa. En aquesta funcionalitat, ha pres part el suport XML un cop ha començat a estar disponible i també disposa de visualització gràfica de les dades cridada des del debugger.

7.2 Principis de disseny

7.2.1 Projecció estructural dels objectes del model

La idea inicial per al disseny del sistema d'activació i passivació XML a CLAM prové del sistema d'activació i passivació als entorns Smalltalk[50]. En aquests entorns, la passivació d'objectes es fa recursivament. El sistema coneix les relacions de composició dels objectes. Activar o passivar un objecte implica activar o passivar els seus components de forma recursiva fins arribar als elements terminals que no contenen altres objectes.

Quan una màquina virtual Smalltalk fa aquesta recursió, la fa, per defecte, basant-se en la informació que pot obtindre de l'objecte gràcies a la capacitat d'introspecció del llenguatge. Smalltalk pot coneixer en temps d'execució la classe, les relacions d'herència de la classe, els atributs que té, els mètodes...

La recursió que fa Smalltalk es basa en un mètode que es defineix per defecte per fer servir la introspecció però que cada classe pot redefinir per establir el seu propi

format. Com que la redefinició es limita al nivell immediatament inferior, i es delega la resta als subcomponents i no cal reimplementar tot.

En resum, Smalltalk proporciona dos mecanismes que minimitzen el codi que ha de implementar un usuari en crear un objecte nou:

- Per un costat, s'ofereix un mecanisme basat en introspecció que dóna una implementació per defecte de la serialització.
- Per un altre costat, es dona la possibilitat de definir un format personalitzat per a una classe definint només el primer nivell de composició i delegant la resta als subcomponents.

En aquest sistema, cal també remarcar l'existència de primitives. Les primitives són aquells elements constructius que impliquen accions directes sobre el destí/origen de la passivació/activació.

Com a primitives podem entendre

- tant objectes fulla que tenen una representació no composada al format,
- com altres accions directes que es puguin fer sobre el medi.

De fet, les primitives són les que, al final construeixen o interpreten el format final.

En resum, s'observa que l'aproximació 'jerarquia i primitives' s'adapta a l'estructura jeràrquica d'XML fent una projecció entre l'estructura del model de dades i la del format resultant. També s'observen, però, certs problemes que cal resoldre i que tenen a veure amb el llenguatge d'implementació (C++).

Com a llenguatge, C++ té una capacitat d'introspecció molt limitada. Els mecanismes d'informació de tipus en temps d'execució que ofereix l'estàndard limiten molt el que es pot arribar a fer, però, a més, les darreres implementacions de l'estàndard no suporten completament aquests mecanismes.

Així doncs, en una primera aproximació, l'únic mecanisme que es pot oferir és el de la redefinició dels mètodes d'activació i passivació. Més endavant es considerarà una alternativa a la introspecció en temps d'execució basada en metaprogramació estàtica en temps de compilació, i es veurà que és possible aplicar-la a la majoria d'objectes de CLAM. Per altres objectes caldrà implementar explícitament el suport de la serialització.

Alguns autors [23] també han intentat aplicar metaprogramació per obtenir serialització automatitzada en C++. Les seves solucions per suplir la introspecció, igual que les nostres, només són aplicables a un domini concret però comparteixen algunes eines.

7.2.2 Múltiples formats

L'aproximació que fa Smalltalk té un inconvenient en sí mateixa que es fa sentir quan tenim en compte els requeriments del sistema. Un d'aquests requeriments del sistema és que el sistema de serialització ha de preveure la seva extensió a d'altres formats que no només XML. No és pas un requeriment que calgui implementar, però, és un requeriment que cal planificar per facilitar-ne futures implementacions.

Un exemple de format que clarament caldria suportar en un futur és SDIF[84]. SDIF és un format estàndard per intercanviar descripcions d'àudio entre diferents aplicacions que es feia servir a les anteriors versions de SMS[41].

Segons la solució anterior, per a cada format que volem implementar, caldria fer un mètode diferent a cada classe. Aquest és un dels principals problemes amb els que ens enfrontem perquè això és molt intrussiu en la interfície dels objectes i afecta, sobretot, als objectes del model que implementa l'usuari.

L'objectiu del disseny és doble: concentrar, a dintre de CLAM, les modificacions que calgui fer per donar suport a un nou format i minimitzar l'esforç de l'usuari final en la creació de nous objectes del model. Cal considerar quin serà el possible escenari en el futur per poder avaluar les forces que hi intervenen i seleccionar la solució que

sembli millor o menys dolenta.

Tenim N objectes del model que han de representar-se amb M formats. Sempre que s'afegeix un format, cal implementar la serialització en aquest format per a N objectes del model, i cada cop que s'afegeix un objecte cal afegir la seva serialització per als M formats.

Aquest problema es pot suavitzar amb diverses solucions que es diferencien pel lloc a on desplacen la gestió del canvi. Per poder valorar el cost de cada solució, cal avaluar primer la probabilitat de cada tipus de modificació.

El nostre domini té les següents característiques:

- Els objectes del model a representar en XML són molt nombrosos. Podem identificar algunes categories:
  - Processaments
  - Dades de processament
  - Configuracions
  - Estructures de control de flux
  - Objectes bàsics
- Part dels objectes del model són propis de la llibreria i una altra gran part seran creats per l'usuari.
- L'usuari crearà, principalment, objectes de les tres primeres categories. Les estructures de control de flux estaran implementades a dintre de la llibreria, i es podran crear de forma puntual objectes bàsics nous.
- Els formats no són tan nombrosos i no es contempla que els usuaris puguin afegeixin nous formats pel seu compte sense modificar la llibreria.

7.2.3 Peculiaritats del format XML

Un cop analitzades les forces que intervenen en el fet de preveure el suport de múltiples formats, en aquest apartat s'analitza què comporta donar suport a un llenguatge concret. Primer es farà un anàlisis de quins són els elements característics de XML i després intentarem extrapolar aquest anàlisi a d'altres formats.

Per emmagatzemar un objecte CLAM en XML, cal determinar un seguit de coses:

1. Cóm projectar l'estructura de l'objecte CLAM en l'estructura XML.
2. Per a cada subobjecte, si és un element, un atribut o contingut pla.
3. En el cas que sigui un atribut o element, quin nom de tag o d'atribut té.
4. Cóm extreure contingut textual d'aquests subobjectes per omplir els atributs i el contingut pla.

Aquestes característiques corresponen amb les primitives del format que s'han mencionat abans.

Si prenem la semàntica de XML Schema com a referència, el fet de si un objecte s'emmagatzema com a element o no i el nom que té, no es quelcom propi de l'objecte (o millor dit del tipus de l'objecte) sinó que es quelcom que es determina al tipus de l'objecte que el conté. De fet el propi tipus de l'objecte XML contingut també el determina l'objecte contenidor però, aquest fet és inherent a la definició en C++ de l'objecte.

Com s'ha explicat a la secció 4.3, un objecte XML es pot veure sovint com a una estructura de C. En una estructura complexa, el nom i el tipus que té un objecte està determinat per l'objecte que l'inclou. És a dir, no té gaire sentit ficar tots els elements `Spectrum` com a element i amb el mateix nom, per exemple *Spectrum*, perquè és l'objecte que el conté – o, si està en el nivell superior, l'aplicació – qui decideix quin nom té depenen de la funció que aquest objecte tingui dintre d'aquest altre objecte.

Per exemple, un Frame de SMS conté diversos espectres: un conté la component harmònica, un altre conté la no harmònica, un altre el residu... En representar això en XML segurament hom vulgui posar a cada espectre un nom més significatiu, no pas el nom del tipus i prou.

En resum, és l'objecte contenidor el que ha de determinar com els continguts s'emmagatzemen.

Tot i així, la forma d'extreure contingut textual d'un objecte sí que és quelcom propi del tipus de l'objecte, però sovint només s'aplica als objectes terminals.

7.3 Mecanisme general

En aquest apartat s'expliquen els mecanismes generals implementats per al suport XML a CLAM. Més endavant s'explicaran per separat els aspectes de la implementació no comuns a tots els objectes, com per exemple, els *dinamic types*, els contenidors STL...

7.3.1 Storages

Storage és una interfície que permet encapsular quelcom que pot emmagatzemar l'estat d'un objecte del model. Hi ha una implementació diferent d'aquesta interfície per a cada format. Per exemple, quan s'emmagatzema un objecte en un XMLStorage, l'objecte queda emmagatzemat en format XML, però, si fem servir un Storage que sigui específic per SDIF hauríem d'obtindre l'estat de l'objecte en format SDIF.

És l'Storage qui fa servir internament les primitives de generació i interpretació del format.

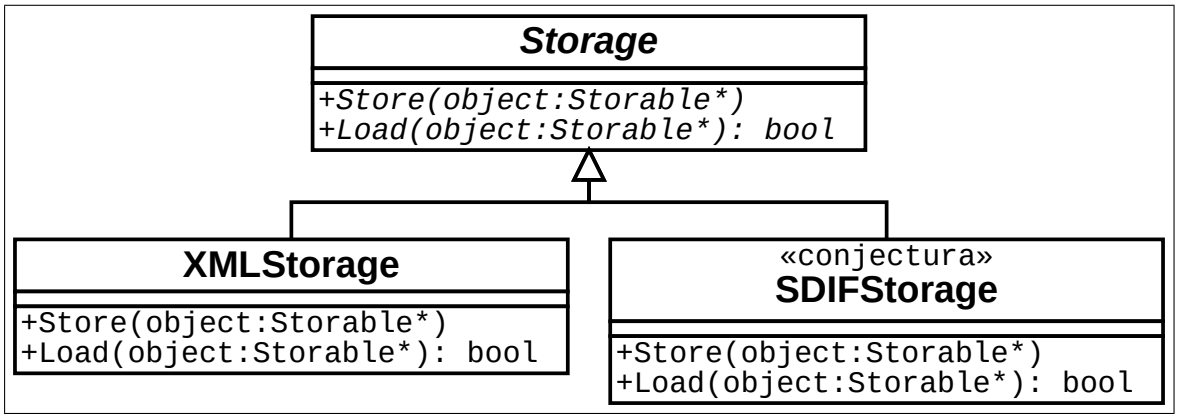

Figura 7.1: Diagrama de classes d'Storage

7.3.2 Storables

Cada `Storage` concret requereix obtindre dels objectes diferent informació segons el format. Per això, cada *Storage* exigeix a l'objecte que implementi l'interfície que li donarà aquesta informació.

Les subinterfícies d'*Storable* defineixen els mètodes que cal implementar per que el corresponent `Storage` pugui obtindre aquesta informació.

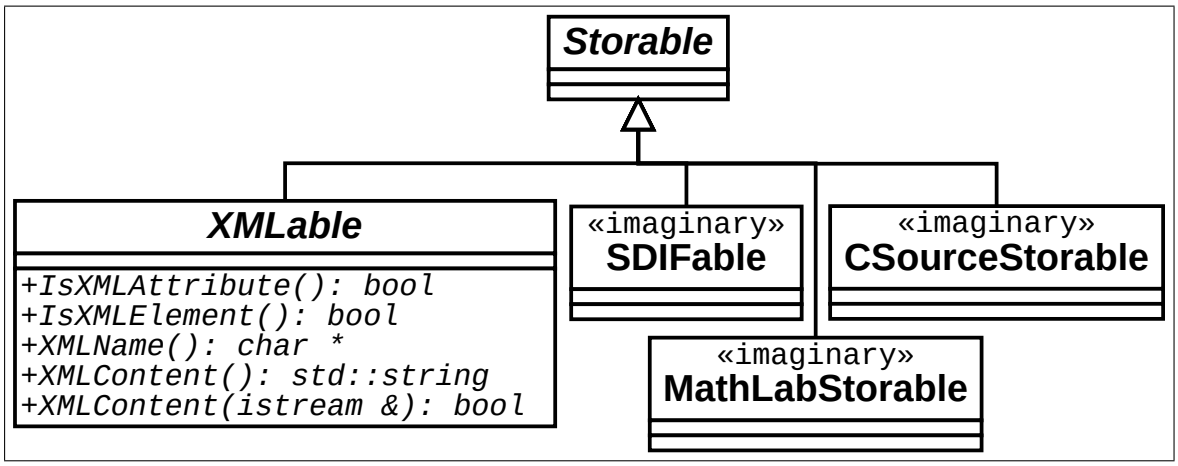

Figura 7.2: Diagrama de classes d'Storable

Per exemple, l'`XMLStorage` requereix dels objectes que s'hi insereixen o s'hi extreuen la

implementació de la interfície `XMLable`. Aquesta ofereix al XMLStorage la informació necessària per conèixer quin tipus d'objecte XML és (si és un atribut, un element o contingut pla), i el nom i com extreure o inserir el contingut planer si s'escau.

7.3.3 Components

Un `Component` és la interfície que ha de implementar tot objecte que vol esdevenir part d'una composició d'una manera coneguda per CLAM. D'aquesta manera CLAM pot aplicar algunes operacions recursives fent servir la interfície de Component sense preocupar-se de les particularitats de l'objecte en concret i de com implementa aquest objecte la composició.

Un `Component` pot incloure objectes o ser inclòs per altres `Component`s formant una estructura jeràrquica. La composició de components determina l'estructura de les dades de l'aplicació, estructura que normalment es projecta en la estructura d'un document XML o d'un altre format.

Després de que un `Storage` accepta un `Storable` com a un vàlid per emmagatzemar, és a dir, comprova que implementa l'interfície adient, comprova si és també un `Component`. Si ho és cridarà als mètodes `StoreOn` o `LoadFrom` que que es defineixen a l'interfície `Component`.

Aquestes funcions són les encarregades de emmagatzemar o carregar recursivament els subobjectes que formen part del `Component` en qüestió.

Aquest mecanismes es veu molt més clar amb un exemple. Imaginem la composició d'objectes representada a la figura 7.3. El diagrama de seqüència de la figura 7.4 representa el conjunt de crides que es desencadenen quan passivem l'objecte `o1` dintre d'un Storage. El mateix procés succedeix quan el que fem és activar l'objecte, però amb els mètodes `LoadFrom` i `Load`.

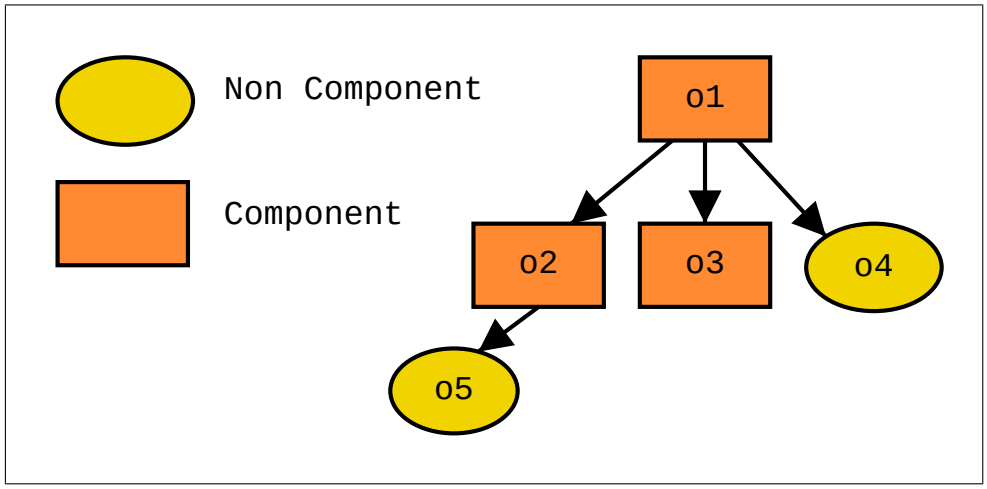

Figura 7.3: Exemple d'estructura de components

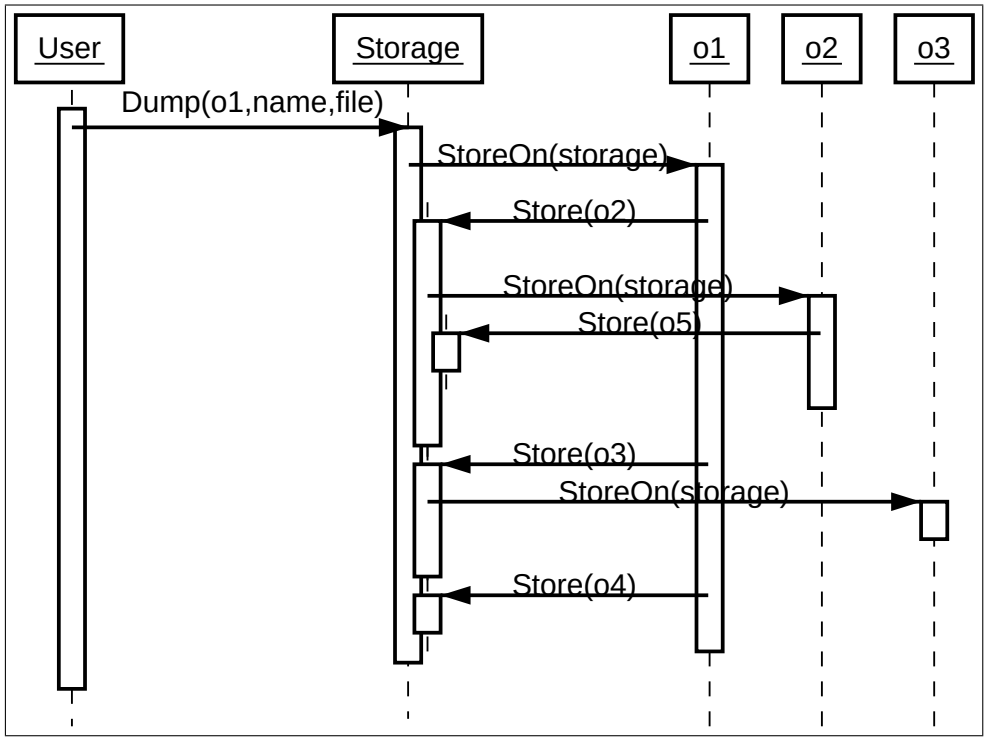

Figura 7.4: Recursió a la sortida amb components

7.3.4 Emmagatzemant XMLables

L'`XMLStorage` requereix dels objectes inserits que implementin la interfície `XMLable`. Aquest apartat és una petita descripció d'aquesta interfície. Aquesta ofereix la informació necessària per conèixer quin tipus d'objecte XML és (si és un atribut, un element o contingut pla), i el nom i/o el contingut pla si s'escau.

El llistat 7.2, esquematitza com es fa servir aquesta informació. Aquesta imple-

```
Storage::Store(Storable*)
    if object is not an XMLable
        return

    if object->IsXMLAttribute()
        store it as attribute using object->XMLContent()
            and object->XMLName()
    else
    if object->IsXMLElement()
        store it as element using object->XMLName()
        if object is a Component
            object.StoreOn(this)
    else
        store it as content using object->XMLContent()
        if object is a Component
            object.StoreOn(this)
```

Llistat 7.2: Pseudocodi de XMLStorage::Store

mentació té algunes implicacions:

- Emmagatzemar objectes que no són XMLables no tenen cap efecte sobre l'*Storage*.
- Un XMLable que sigui atribut no fa recursió pels seus subcomponents encara que sigui Component.
- Els subcomponents d'un objecte que sigui contingut s'emmagatzemen com a germans XML seus i no pas com a fills.

Ho podem veure en el següent exemple: l'estructura de components XMLables que es descriu a la taula 7.1, es formatejarà com indica el llistat 7.3

Nom	Tipus	Contingut
• Document	Element	' '
• A	Atribut	'ContingutAtribut'
• AA	Atribut	'Contingut-Atribut.Atribut'
• AE	Element	'Contingut-Atribut.Element'
• AT	Text	'Contingut-Atribut.TextPlaner'
• E	Element	'Contingut-Element'
• EA	Atribut	'Contingut-Element.Atribut'
• EE	Element	'Contingut-Element.Element'
• ET	Text	'Contingut-Element.TextPlaner'
• T	Text	'Contingut-TextPlaner'
• TA	Atribut	'Contingut-TextPlaner.Atribut'
• TE	Element	'Contingut-TextPlaner.Element'
• TT	Text	'Contingut-TextPlaner.TextPlaner'

Taula 7.1: Exemple: Estructura de components XMLables

```
<Document A='Contingut-Atribut' TA='Contingut-TextPlaner.Attribut'>
    <E EA='Contingut-Element.Atribut'>
        Contingut-Element
        <EE>
            Contingut-Element.Element
        </EE>
    Contingut-TextPlaner
    <TE>
        Contingut-TextPlaner.Element
    </TE>
    Contingut-TextPlaner.TextPlaner
</Document>
```

Llistat 7.3: Efecte dels tipus de XMLable al format

7.3.5 Adaptadors XML

Implementar la interfície `XMLable` per tot objecte que es vulgui representar en XML, no és una bona solució per un seguit de raons:

- Cap tipus nadiu del C no pot implementar-ho perquè no són classes redefinibles.
- Moltes classes no voldran tenir la càrrega extra que representa implementar aquesta interfície.
- Algunes classes externes simplement no poden ser modificades.

A més, per raons que ja han estat comentades, no convé transportar als objectes del model la complexitat associada amb la diversitat dels formats. Sobretot quan molts d'aquests objectes seran objectes d'usuari.

Per això, no s'ha volgut implementar la interfície XMLable dins dels objectes del model. Si apareix la interficie SDIFable caldria implementar els nous mètodes a tots els objectes del model, inclosos els d'usuari.

La solució és oferir objectes adaptadors que serveixen per adaptar els objectes del model a aquesta interfície.

El procés d'adaptació el pot fer el component que conté els objectes en el mateix mètode `StoreOn` o `LoadFrom`. Si la fa el Component podem seguir el criteri de XML-Schema: el nom, el tipus de node, i el tipus de dada de cada objecte està determinat pel tipus de dada del seu contenidor i no pel tipus de dada del propi objecte.

S'ha implementat als adaptadors una interfície tal que el component que els fa servir pot especificar, en el constructor de l'adaptador, el nom i el tipus d'element XML a més d'especificar i caracteritzar l'objecte adaptat.

La figura 7.5 representa el diagrama de classes dels diferents adaptadors. Les funcionalitats comuns com ara guardar el nom i el tipus de node XML s'implementen en una superclasse comuna. Les funcionalitats pròpies de cada tipus d'adaptació s'implementen als adaptadors concrets que en deriven. Les funcionalitats que s'especialitzen

estan relacionades sobretot amb el contingut pla, que depén totalment de l'objecte adaptat, i la forma de respondre a les característiques de component. Els adaptadors concrets s'expliquen a continuació.

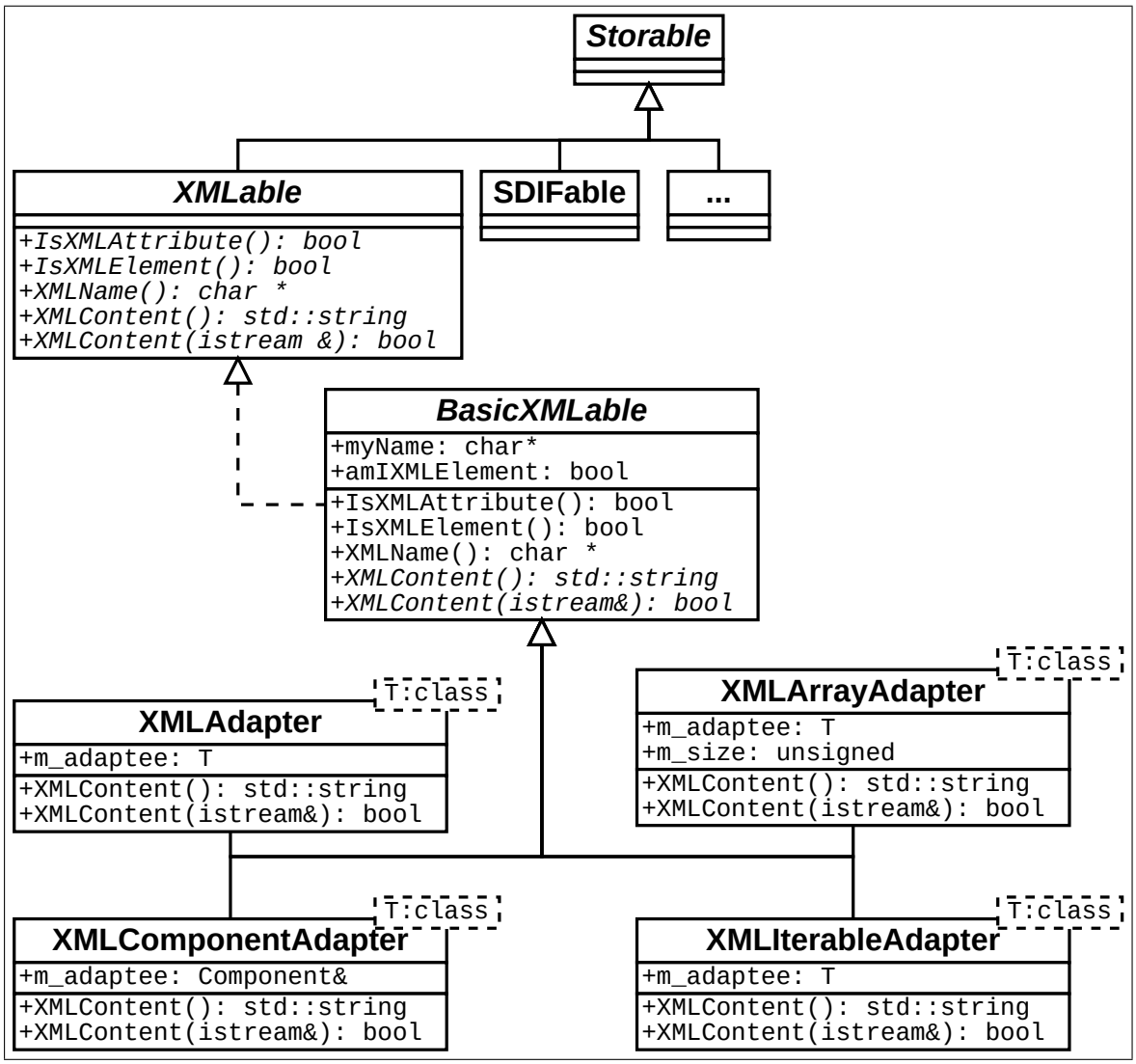

Figura 7.5: Jerarquia de clases dels adaptadors XML

Adaptadors de tipus simples

A CLAM es considera un tipus simple, aquell que implementa els operadors d'inserció i extracció (<< i >>) amb `ostream`s de C++. La llibreria estàndard de C++ defineix aquest operadors per la majoria de tipus bàsics de C (`int`, `float`, `char`, `char*`...). CLAM i els usuari també poden definir-los per els seus propis objectes.

Aprofitant això, CLAM ofereix un adaptador XML, `XMLAdapter`, *templatitzat* pel tipus de l'objecte adaptat, que fa servir aquests operadors per extreure el contingut XML. Fent servir aquest adaptador, fins i tot molts objectes que no són CLAM poden esdevenir XMLables.

Aquesta solució té alguns problemes greus amb les cadenes de text, ja siguin punters a `char` o `std::string`. Els operadors d'inserció i extracció de les cadenes de text no són simètrics. La inserció d'una cadena es fa sempre de forma integra. En canvi la extracció es fa en tokens separats per espais. Per això cal advertir a l'usuari per que sigui concient d'aquest fet i no insireixi cadenes amb espais.

Més endavant es considerarà algun tipus especial que serveixi per emmagatzemar i recuperar text complexe però hauria d'estar delimitat de forma especial per limitar les ambigüitats.

Adaptadors d'objectes simples adjacents

Un dels adaptadors que CLAM ofereix és un adaptador especialitzat per objectes simples adjacents en memòria, és a dir, les arrays, en el sentit tradicional de C, d'objectes simples.

Aquest adaptador és un template anomenat `XMLArrayAdapter` que a més de passar-li el punter on comença l'array, també cal dir-li el nombre d'elements. El contingut extret per aquest adaptador és la concatenació de les cadenes que extreuria el adaptador simple separades per espais.

En la càrrega, cal saber aquest nombre d'elements abans de construir l'adaptador, i

si el nombre d'elements depén del contingut del document XML, caldria assegurar-se de que la informació de la mida de l'array arriba abans que la càrrega de l'array.

També cal tenir present, de cara a l'activació, que la separació d'un espai sigui suficient per identificar els elements.

Adaptadors de Components

L'adaptador `XMLComponentAdapter` és un adaptador per a components. Aquest adaptador cal que sigui, ell també un component. D'aquesta manera, quan el `Storage` comprova si ha de demanar la recursió, veu que el XMLable és un component i crida al `StoreOn` o al `LoadFrom` del adaptador.

L'adaptador implementa aquests dos mètodes, propis dels components, transmetent la crida al mètode anàleg de l'objecte que adapta.

Adaptadors de contenidors heterogenis

Un altre tipus d'adaptador XML està orientat a contenidors que segueixin les interfícies dels contenidors STL. Aquest adaptador es diu `XMLContainerAdapter`.

Aquests contenidors són una mica més complexos que els adaptadors d'arrays perquè permeten, fent servir algunes tècniques de metaprogramació, adaptar no només contenidors de tipus bàsics sinó també contenidors de components i sap ignorar els contenidors d'elements que no són ni components ni tipus bàsics.

Fa servir el concepte genèric iteradors de la STL (vegeu apartat 6.3.5) per obtenir un algorisme que serialitzi tots els elements del contenidor.

Com ara podem tenir diversos tipus d'elements, cal que fem una selecció de mètode estàtica (vegeu apartat 6.3.4) basant-nos en dos paràmetres. El mecanisme concret és el mateix que després explicarem a l'apartat 7.4.5.

7.3.6 Manegament del DOM

La implementació interna de la passivació XML és senzilla. Les primitives de format són les de construcció del DOM. Cal mantenir un context posicionat a un element dintre del DOM. Els nous nodes XML que arriben s'afegeixen dintre de l'element de context i, cada cop que el node introduït és un nou element, el context es canvia per que apunti a aquest nou element fins que la passivació dels nodes que conté hagi acabat i es restauri l'antic element de context.

El mètode utilitzat té una part negativa i és que en emmagatzemar un document gran, cal mantenir el model DOM a memòria. Una alternativa hagués estat no construir el arbre DOM i escriure el document formatejat al medi directament a mida que arriben els nodes. El problema amb aquesta aproximació és que els atributs tot i que s'inserissin al final de tot cal formatar-los abans que els altres nodes, i caldria retenir la sortida.

De cara a l'activació, el principal problema és que els nodes adjacents de text planer, tot i que a la passivació fossin nodes independents, a l'activació es fusionen en un únic node.

Tot i que, amb un XML-Schema prou restringit, identificar diferents nodes de text adjacents seria possible, actualment, el suport d'Schemes de les llibreries que en tenen es limita a validació i no es fa servir pas com a ajuda al parsing ni transcendeix informació al arbre DOM construït.

En tot cas, cal tenir cura amb la implementació dels operadors d'inserció i extracció per evitar consumir més entrada de la que cal.

Es per això que, tot i que donem molta flexibilitat al format XML que es pot generar amb CLAM, cal fer que sigui possible llegir-los. Simplement, el format resultant pot ser ambigu o dificultar la forma de llegir. XML no ens allibera d'introduir ambigüitats en el format.

7.4 Dynamic Types

7.4.1 Concepte i utilitat

A CLAM sovint ens trobem classes de la llibreria que treballen amb un nombre molt gran d'atributs i, que al mateix temps, cada instància d'aquestes classes només en fan servir a cada execució un subconjunt reduït. Per exemple, un descriptor de segment d'àudio que només manega la part dels descriptors que és necessària per a un processament en concret, o els objectes de configuració que no necessiten especificar tots els paràmetres quan existeixen valors per defecte, o s'ofereixen paràmetres redundants entre ells.

Els *dynamic types* són una solució que permet optimitzar la memòria ocupada per aquest tipus objecte. Un tipus dinàmic defineix un seguit de atributs dinàmics que en temps d'execució es poden instanciar i desinstanciar eliminant la seva marca a memòria. Estan definits de tal forma que amb excepció del protocol d'instanciació, semblen atributs privats accedits per accessors.

Els dynamic types també implementen sense intervenció de l'usuari la interfície de `Component` que permet a CLAM conèixer la composició d'objectes i actuar en conseqüència.

La relació dels tipus dinàmics amb XML és que les mateixes capacitats que permeten instanciar i desinstanciar atributs possibiliten, sense cost afegit, una certa capacitat d'instrospecció que és molt útil per donar les implementacions per defecte de la serialització que donaven altres llenguatges com ara Smalltalk i Java.

Cal aclarar que *dynamic types* és el nom que reben a CLAM i no te res a veure amb el que alguns autors[58] anomenen tipus dinàmics: definicions de tipus que es poden crear i manipular en temps d'execució. De fet el que és dinàmic als *dynamic types*, són els atributs, no pas el tipus en sí.

Els *dynamic types* són una part molt central de CLAM, perquè es fan servir per representar tots els objectes de dades de processament i els objectes de configuració.

CLAM ha patit diversos processos d'optimització com a fruit del seu ús a projectes basats en sistemes de temps real. En ser un element tan central de la llibreria, els *dynamic types* han estat l'objecte de moltes d'aquestes optimitzacions agressives, i el seu ús no es considera cap perduda de eficiència, fins i tot en alguns casos facilita algunes optimitzacions addicionals.

Els dynamic types ofereixen dos avantatges de cara a implementar la serialització. Per un costat, la instanciació i desinstanciació d'atributs requereix mantenir en temps d'execució informació sobre els atributs, compartida per tots els objectes del mateix tipus que ens pot ser molt útil per emmagatzemar els atributs dinàmics de la forma adecuada.

Per un altre costat, part de la complexitat de la implementació dels dynamic types és responsabilitat de les mateixes classes concretes. Aquesta responsabilitat s'ha ocultat als programadors amb l'ús de macros que generen les implementacions d'alguns mètodes. Podem afegir funcionalitat a la ja proporcionada per les macros agafant la informació proporcionada com a paràmetres de les macros.

7.4.2 Estructura interna

Els atributs dinàmics no es corresponen amb atributs normals de C++. Estan situats a un bloc de memòria que gestiona el dynamic type. Els objectes que representen els atributs es creen i es destrueixen mitjançant els operadors de construcció (new) i destrucció (delete) sobre l'emmagatzemament ja disponible a aquest bloc de memòria.

Quan instanciem o desinstanciem un atribut es marca aquest atribut a una taula d'instanciacions d'atributs, pròpia de cada objecte. Cal demanar explícitament a l'objecte que actualitzi les seves dades per reajustar l'espai disponible al bloc de memòria dels atributs. Així optimitzem les reallocacions de la taula de dades.

En principi, la taula d'instanciacions és pròpia de cada objecte però, es pot optimitzar tot compartint-la entre diversos objectes que tenen la mateixa matriu o prototipus. La taula d'instanciacions es comparteix amb el criteri de que quan un objecte intenta

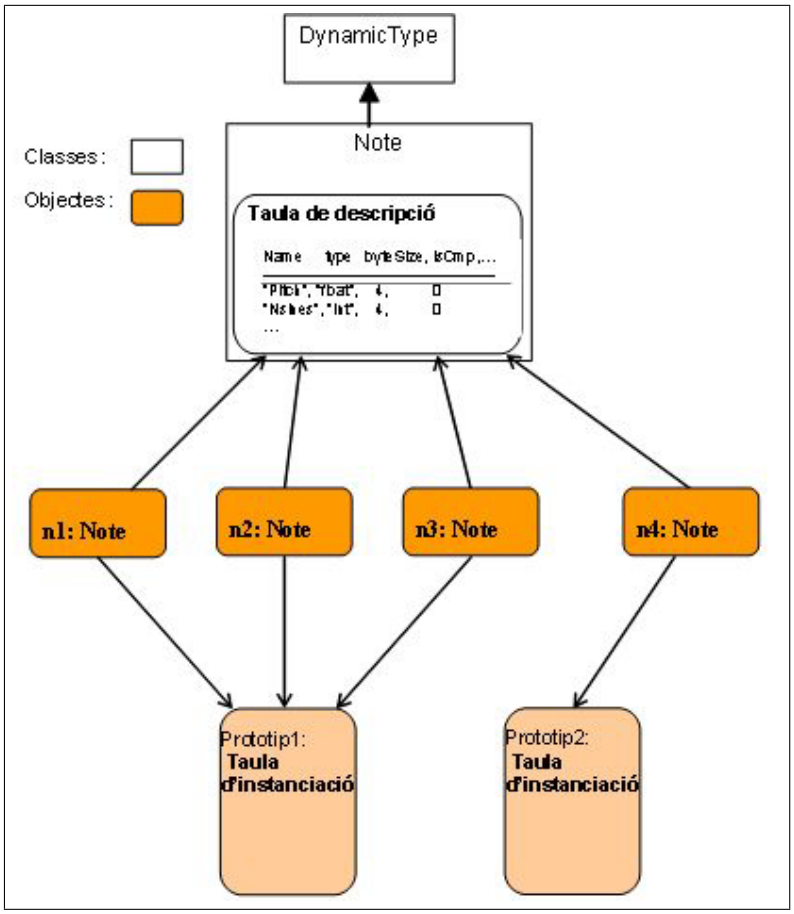

Figura 7.6: Diagrama de classes d'Storable

modificar-la, llavors ha de construir la seva pròpia taula d'instanciacions.

A més de la taula d'instanciacions, cada tipus de DT té una taula compartida per totes les intàncies d'aquest tipus concret on hi ha informació general sobre els atributs disponibles (tipus, nom, característiques...).

En resum:

- Hi ha una taula compartida entre totes les instàncies amb informació comuna de descripció dels atributs dinàmics: la **taula de descripció**.
- Hi ha una taula compartida entre les instàncies provinents d'un mateix prototipus que conté informació sobre l'estat d'instanciació de cada atribut: la **taula d'instanciació**.
- A cada objecte hi ha una taula de dades que conté el contingut dels atributs instanciats per cada objecte: la **taula de dades**.

7.4.3 Primera implementació

La primera implementació de la serialització als dynamic types només considerava la passivació XML i encara no feia activació. Feia servir la informació en temps d'execució que hi ha disponible a la taula compartida de descripció dels atributs.

A la taula de descripció d'atributs hi havia la següent informació útil de cara a implementar l'`StoreOn`: Una cadena amb el nom de l'atribut, una altra cadena amb el nom del tipus i un quants valors booleans que indicaven, entre altres coses, si l'atribut era un Component.

En aquesta primera implementació, la taula de descripció s'omplia a mida que els atributs s'anaven instanciant. Cada cop que s'instanciava un atribut, primer es mirava si l'atribut havia estat informat a la taula de descripció, i si no, llavors s'informava.

La taula de descripció havia de omplir-se a parts des dels mètodes d'instanciació dels atributs concrets perquè els constructors i inicialitzadors s'expandien a la macro

que expandeix els membres comuns i no tenen accessible la informació dels atributs. Aquesta informació només està accessible a les macros que defineixen cada atribut.

```
class CustomDyn : public CLAM::DynamicType
{
public:
    REG_NUM_ATTR(CustomDyn, 5);
    REGISTER(0, MyInt, int);
    REGISTER(1, MySpectrum, CLAM::Spectrum);
    REGISTER(2, MyAudio, CLAM::Audio);
public:
    void Init() {
        AddMyInt();      // Instanciem MyInt
        AddMySpectrum(); // Instanciem MySpectrum
        RemoveMyAudio(); // Desinstanciem MyAudio
        UpdateData();    // Sincronitzem les dades
    }
};
```

Llistat 7.4: Definició d'un tipus dinàmic (antic estil)

Per fer la detecció de si el tipus és un component es va fer servir una tècnica de static dispatch, sobrecarregant un paràmetre d'una funció per a punter a component i punter a void. S'enviava en el lloc del paràmetre, un valor nul convertit en punter al tipus de l'atribut.

Per fer la passivació es recorria la taula de descripció del primer a l'últim atribut. Si l'atribut era un Component, es feia un *cast* a Component de la posició de memòria corresponent i es feia servir un `XMLComponentAdapter` per fer l'Store. Si no era Component, llavors es comparava la cadena del tipus amb les cadenes d'alguns tipus bàsics comuns i si coincidia es feia el *cast* i es feia servir un `XMLAdapter` instanciat pel tipus concret.

S'observa que la forma de detectar els tipus bàsics és molt feble. No és tolerant a noms de tipus sinònims i força a canviar la implementació de l'StoreOn cada cop que afegim un nou tipus bàsic.

Quan es va plantejar el fet d'implementar l'activació XML, es va veure que aquesta implementació era molt limitant i es varen introduir tàctiques inspirades en la

metaprogramació i en la programació genèrica per reconstruir els tipus dinàmics.

7.4.4 Mètodes encadenats

Vam observar que no era necessari deduir el tipus a partir del seu nom ni guardar enlloc el nom de l'atribut, per fer un mètode que fes la passivació o activació d'un atribut concret, si aquest mètode s'expandia amb la macro d'aquest atribut. Tenim el tipus que podem fer servir directament al codi, i tenim el nom de l'atribut que podem fer servir per cridar les funcions que ens donen informació sobre ell, sense accedir a cap taula de descripció.

El problema més important era com fer que un mètode expandit per la macro comuna, pogués cridar a cadascun dels mètodes expandits per les macros d'atribut. La forma que fèiem servir per que les macros d'atribut no col·lisionessin, era fer que el nom de l'atribut formés part dels noms dels mètodes: `AddMyInt`, `RemoveMyInt`, `GetMyInt`, `SetMyInt`...

Així doncs, des de la macro comuna no sabíem quin és el nom del mètode corresponent de cada atribut.

En C++ una altra forma de fer que dos mètodes no col·lisionin encara que tinguin el mateix nom és mitjançant la sobrecàrrega. Si els tipus dels paràmetres són diferents, els mètodes són diferents. No és vàlid sobrecarregar pel tipus de l'atribut donat que aquest es pot repetir.

La solució ve de la mà dels templates i s'inspira en les tècniques de metaprogramació. Cada atribut té associat un enter que indica la posició en la taula de descripció. Donat que els enters serveixen com a paràmetres template podem fer que el tipus que necessitem per la sobrecàrrega sigui un template instanciat per aquest enter.

Això ens deslliga dels noms, però com es pot cridar des del mètode comú tots i cadascun dels mètodes per a cada atribut? Un altre cop es poden fer servir les tècniques de metaprogramació: Des del mètode comú, es crida al mètode sobrecarregat per `AttributePosition<0>`. A partir d'aquí cada mètode de la cadena, després de fer la

tasca pel seu atribut, cridarà al mètode següent de la cadena que serà el sobrecarregat per `AttributePosition<Index+1>`. La cadena finalitza quan es crida al mètode sobrecarregat per `AttributePosition<N>` on N és el nombre d'atributs. Aquest darrer mètode es defineix com a un mètode buit i s'expandeix a dintre de la macro comuna.

```
#define DYN_ATTRIBUTE(N,ACCESS,TYPE,NAME) \
ACCESS: \
    void Do##NAME() { \
        /* Here we have available all the attribute information */ \
        std::cout << N << "\t"#TYPE"\t"#NAME << std::endl; \
    } \
private: \
    void DoChainedAttr(AttributePosition<N>*) { \
        Do##NAME(); \
        DoChainedAttr((AttributePosition<(N)+1>*)NULL); \
    } \
```

Llistat 7.5: Mecanisme d'encadenació de mètodes: Macro d'atribut

Totes les crides que es fan amb l'encadenament de mètodes, en el fons no tenen cap penalització en l'eficiència. De fet, aquestes crides són inline i la crida a la funció que inicia la cadena equival a posar les accions corresponents a cada atribut, una darrera de l'altra.

El que sí que era problemàtic amb els mètodes encadenats eren els errors que donava el compilador quan hi havia alguna inconsistència en els índexs que es posaven als atributs.

Vam fer servir un xic més de metaprogramació per generar errors de compilació semblants als que es comentaven a l'apartat de programació genèrica. Les definicions per defecte dels chained methods es fan servir per detectar quan s'havia produït una inconsistència en els índexs i, amb algunes tècniques de metaprogramació que es veuen al llistats 7.5 i 7.6 es pot informar a l'usuari de quin tipus d'error és (un índex repetit, el·lidit o fora de rang) i quin ha sigut l'índex erroni.

D'aquesta manera, es va aconseguir convertir quelcom que era una disfunció a les noves macros respecte a les antigues (apareixien errors dificils de llegir) en una funcionalitat nova (comprovació de consistència en els índexs d'atribut). Els chained methods han

```
#define DYNAMIC_TYPE(CLASS_NAME,N) \
public: \
    enum { eNumAttr= N }; \
public: \
    /** Do all Dynamic Attributes */ \
    void DoAll () { \
        DoChainedAttr((AttributePosition<0>*)NULL); \
    } \
private: \
    /** Method chain terminator */ \
    void DoChainedAttr (AttributePosition<N>*) { \
    } \
private: \
    /** Undefined link for the Do method chain (Do) */ \
    template <unsigned int NAttrib> \
    void DoChainedAttr (AttributePosition<NAttrib>*a) { \
        CheckAttribute ((AttributePosition<NAttrib>::InboundsCheck*)NULL, \
                        (AttributePosition<NAttrib>*)NULL); \
    }\
private: \
    template <unsigned int NAttrib> \
    class AttributePosition
            : public DynamicType::AttributePositionBase<NAttrib> { \
        public: \
            typedef StaticBool<!(NAttrib>=N)> InboundsCheck; \
    }; \
    /** Instantiated whenever a Attribute number is out of range.
     * Gives a compilation error message.
     */ \
    template <unsigned int NAttrib> \
    void CheckAttribute (StaticFalse*inRange,AttributePosition<NAttrib>*a) { \
        AttributePosition<(NAttrib)-1>* previous; \
        previous->CompilationError_AttributePositionOutOfBounds(); \
    }\
    /**
     * Instantiated whenever a Attribute number is left.
     * Gives a compilation error message.
     */ \
    template <unsigned int NAttrib> \
    void CheckAttribute (StaticTrue*inRange,AttributePosition<NAttrib>*a) { \
        a->CompilationError_AttributeNotDefined(); \
    }\
```

Llistat 7.6: Mecanisme d'encadenació de mètodes: Macro de classe

aportat altres millores als dynamic types, donat que el mateix entrebanc que ens havien trobat per la serialització, ja havia obstaculitzat algunes altres funcionalitats.

Per exemple, ara la taula de descripció s'omple des dels constructors, hi ha mètodes per instanciar i desinstanciar tots els atributs i s'han fet algunes optimitzacions en la forma de reservar espai a la taula de dades.

Els llistats 7.5 i 7.6 no són implementacions complertes però, il·lustren tot el que s'ha explicat abans respecte l'encadenament de mètodes i la detecció d'errors amb una hipotètica cadena de mètodes `DoAll` que crida els corresponents `Do`*`X`* on la X és el nom de cadascun dels atributs.

7.4.5 Selecció de l'adaptador

Un altre detall d'implementació, és com les noves macros seleccionen quin adaptador han de fer servir. La distinció entre component i no component es fa de forma molt semblant a com es feia a la primera versió de les macros: sobrecarregant amb un parametre que pot ser punter a `void` o punter a `Component`.

La diferència està en la forma de detectar quins són els tipus bàsics amb els que es pot fer servir un `XMLStaticAdapter`. Es fa servir una solució semblant a les interfícies estàtiques de McNamara explicades a l'apartat 6.3.4. McNamara, explicitava el fet de que un tipus modelava un concepte per poder, després, fer comprovacions i prendre decissions en en temps de compilació.

En la seva solució, la explicitació de la petanyença a un concepte, es feia dins de la definició de la classe. En el cas que ens ocupa, està clar que això no es pot fer així perquè molts dels tipus que hauran de modelar el concepte són tipus elementals de C o tipus externs que no podem tocar.

L'aproximació que he implementat és explicitar que el tipus modela el concepte, fora de la declaracio del tipus. El que faig servir per fer aquesta declaració és una classe template amb un metavalor booleà (de fet un `typedef`). En la definició per defecte del template aquest valor (tipus) es defineix com a `metafals`. Per dir que un tipus

modela un concepte, només cal especialitzar el template per al tipus i definint aquest typedef com a `metatrue`.

Amb aquest metavalor podem fer una selecció estàtica entre dos mètodes sobrecarregats, un que fa servir l'adaptador i un altre que ignora l'atribut.

La solució es podria millorar tot fent, a més, una comprovació estructural del concepte com proposa McNamara en la seva solució híbrida.

7.4.6 Personalització

Els chained methods ens permeten oferir als usuaris algunes funcions que faciliten la personalització del format per part de l'usuari, respecte a com ho haurien de fer amb els dynamic types.

```
class ConcreteDT : public CLAM::DynamicType
{
public:
    DYNAMIC_TYPE(ConcreteDT, 5);
    DYN_ATTRIBUTE      (0, public, DummyComponent, MyComponent);
    DYN_ATTRIBUTE      (1, public, Array<Complex>, MyArray);
    DYN_ATTRIBUTE      (2, public, FooDTClass, MyDynType);
    DYN_CONTAINER_ATTRIBUTE(3, public, std::list<int>, MyList);
    DYN_ATTRIBUTE      (4, public, int, MyInt);
public:
    virtual ~SuperDyn() {}
protected:
    void DefaultInit()
    {
        AddMyDyn();
        AddMyA();
        UpdateData();
    }
// Some non dynamic attributes
private:
    FooComponent mExtraNonDynamicAttribute;
};
```

Llistat 7.7: Serialitzant atributs no dinàmics

Primer de tot, els mètodes `StoreOn` i `LoadFrom` es poder redefinir a les subclasses concretes de DynamicType donat que es defineix a la mateixa classe DynamicType. Aquesta definició el que fa es cridar una funció virtual que es defineix a les subclasses i enceta chained method que fa la serialització de cadascun dels atributs dinàmics.

Una forma senzilla d'afegir la serialització d'atributs que no siguin dinàmics és redefinir l'`StoreOn` i el `LoadFrom` perquè serialitzin aquests atributs i després cridin al chained method corresponent per serialitzar els atributs dinàmics.

```
void ConcreteDT::StoreOn(CLAM::Storage &amp; storage) {

// Store a temporary object in the firts place
    CLAM::XMLAdapter<char*> adapter1("Addedcontent", "Added", false);
    storage.Store(&adapter1);

// Call the default implementation
    StoreAllDynAttributes();

// Store a non dynamic attribute member
    CLAM::XMLComponentAdapter adapter2(mExtraNonDynamicAttribute,
        , "ExtraNonDynamic", true);
    storage.Store(&adapter2);

}

void ConcreteDT::LoadOn(CLAM::Storage &amp; storage) {

// std::string is not vulnerable to buffer overflows on loading
    std::string foo; // A temp
    CLAM::XMLAdapter<std::string> adapter1(foo, "Added", false);
    storage.Load(&adapter1);

    LoadAllDynAttributes();

    CLAM::XMLComponentAdapter adapter2(mExtraNonDynamicAttribute,
        , "ExtraNonDynamic", true);
    storage.Load(&adapter2);
}
```

Llistat 7.8: Afegint atributs no dinàmics a la serialització

Si el que es vol personalitzar és la forma en que es serialitzen els mateixos atributs dinàmics, podem fer servir els metodes `StoreX` i `LoadX` (on X és el nom de l'atribut)

```
void ConcreteDT::StoreOn(CLAM::Storage & storage) {

    StoreMyDummyComponent(storage);
    StoreMyArray(storage);
    StoreMyDynType(storage);
    StoreMyList(storage);

    // MyInt is stored as an attribute (the default is element
    // and with a different name ('Size').

    if (HasMyInt()) {
        CLAM::XMLStaticAdapter adapter(GetMyInt(), "Size", false);
        storage.Store(&adapter);
    }
}
void ConcreteDT::LoadOn(CLAM::Storage & storage) {
    // First of all asure that all attributes are instantiated
    AddAll()
    UpdateData();
    // Then load them
    LoadMyDummyComponent(storage);
    LoadMyArray(storage);
    LoadMyDynType(storage);
    LoadMyList(storage);

    // MyInt is loaded as an attribute (the default is element
    // and with a different name ('Size').

    CLAM::XMLAdapter<int> adapter(GetMyInt(), "Size", false);
    if (!storage.Load(&adapter)) {
        RemoveMyInt();
    }
}
```

Llistat 7.9: Modificant la serialització d'atributs dinàmics

que es criden des de la cadena, per serialitzar atributs que es continuen serialitzant de la forma estàndard (element i amb el nom de l'atribut de nom XML) i implementar manualment amb adaptadors aquells es serialitzin de forma diferent.

7.4.7 Referències a objectes externs

Actualment, els dynamic types no permeten atributs que siguin punters. En el passat, els atributs punters i la seva gestió havien estat una font de maldecaps.

En el reanàlisi dels dynamic types que es va fer amb la introducció dels mètodes encadenats, es va determinar que només es coneixien dos casos en que l'ús de punters en un dynamic type eren insalvables: Quan voliem fer servir un atribut polimòrfic, i quan voliem mantenir una referència a un objecte del qual el dynamic type no era propietari, és a dir, que l'objecte no estava contingut a la taula de dades.

Ambdós casos tenen les seves pròpies complexitats i com a conseqüència solucions molt especialitzades. En aquest apartat i en el següent s'han recollit aquestes solucions des del punt de vista de la serialització en XML. Cal remarcar que cap d'aquestes solucions s'ha implementat però, que van ser objecte d'un anàlisi de viabilitat previ, necessari per conèixer si es podria obviar els punters en la primera fase de la implementació.

En el cas de les referències externes, la serialització ha de mantenir en el document XML referències creuades. La forma normal de fer-ho en un document XML és amb identificadors i referències a aquests identificadors. Una altra forma alternativa és amb expressions XPath però, es va descartar donat la complexitat de resoldre l'expressió corresponent a partir del model de dades.

La solució de les referències sembla més senzilla però, no està exempta de complexitats. Suposem en tot cas que les referències són internes dintre d'un mateix document, és a dir, que una referència no resolta dintre del document es considera un error de serialització.

L'activació XML és molt senzilla, només cal tenir una taula amb la projecció de

referències XML cap els punters als objectes referits. La taula s'omple a mida que es carreguen els objectes amb identificador. En el cas de que es carregui una referència cap a un objecte que encara no s'ha activat, llavors el punter s'afegeix a una llista d'espera per aquest identificador concret.

La passivació afegeix un xic més de complicació derivada del fet que no sabem quin seran els objectes referits, i, en conseqüència, no sabem quins són els objectes que necessiten passivar un identificador. Aquest problema té dos solucions directes:

- Fer una neteja a posteriori d'identificadors no referits a l'arbre DOM. Aquesta solució és simple però empitjora els ja desorbitats requeriments de memòria que té el DOM.
- Fer un primer recorregut lleuger (sense construir el DOM) per recopilar les possibles referències.

7.4.8 Atributs dinàmics polimòrfics

L'altre cas d'ús dels punters a dintre d'un dynamic type són els objectes polimòrfics. El que vull expressar amb *objectes polimòrfics* és que el lloc de l'objecte polimòrfic el pot ocupar un objecte que pertanyi a un tipus de dades dintre d'un conjunt de tipus de dades possibles.

El problema amb els objectes polimòrfics és que si bé normalment, quan activem un objecte, el que fem és passar-li un objecte verge al Storage per que l'ompli, ara no sabem quin objecte verge hem de passar-li.

Els patrons de disseny de creació d'objectes ens ajuden en aquest cas. La aproximació ha de ser una factoria que, a partir, d'un identificador de tipus crei l'objecte.

La mateixa especificació XML-Schema ens coherciona a adoptar aquesta solució quan defineix els seus tipus abstractes, que no són més que els nostres objectes polimòrfics. XML-Schema obliga a aquells objectes XML que ocupin una posició polimòrfica a especificar el tipus XML-Schema concret al que pertanyen.

7.5 Altres objectes

De banda de els dynamic types hi ha tota una sèrie de tipus que necessiten tenir implementar la seva serialització. Cal modelar tipus com `Complex`, `Polar`, `Point`, `Pixel`... com a tipus bàsics. Altres com la BPF, que representa l'especificació d'una funció mitjançant interpolació d'un seguit de punts, es modelen com a components.

Hi ha tot un seguit de tipus que necessiten quelcom més d'explicació.

7.5.1 Enumeracions i flags simbòlics

Una enumeració és un tipus de dades tal que les seves instàncies poden adoptar un valor simbólic d'un conjunt de valors disponibles. Tot i que el valor sigui simbòlic, normalment es representa per un valor enter que indica quin dels valors simbòlics adopta. Al llenguatge C++ les enumeracións es representen amb una estructura de dades anomenada `enum` que tenen aquesta equivalència numèrica en temps d'execució i que permet fer servir els símbols en el codi, però, només en el codi.

Un conjunt de flags és un altre tipus de dada que representa un conjunt de valors que poden estar actius o no. En una dada, cada valor o símbol es pot activar o desactivar de forma independent als altres valors. En C++ es pot fer servir els bits d'un valor enter qualsevol per representar aquest conjunt, però, la STL ofereix una abstracció més flexible, el `bitset` que facilita les referències als valors individuals dels flags com si fossin valors booleans independents.

Una qüestió clau perquè la sortida XML quedi molt més llegible és que les enumeracions i els conjunts de valors booleans tinguin representació, no pas numèrica, sinó simbòlica. És a dir, que si tenim una enumeració com ara:

```
enum EScale {Linear, Logarithmic};
```

La seva representació al fitxer no siguin els nombres enters amb els que es representa aquesta enumeració en temps d'execució, sinó els mateixos símbols `Linear` i

`Logarithmic`.

El mateix passa amb els conjunts de valors booleans (flags) on cada valor té un significat. No volem obtenir la representació numèrica de la concatenació de bits sinó saber quins valors simbòlics estan actius o inactius.

Tot plegat tampoc interessa perdre la representació compacta i optimitzada computacionalment que en fa el C++. Així doncs CLAM ofereix una solució de compromís per donar-ne accés simbòlic a flags i enumeracions mantenint la representació C++ a memòria.

La solució és crear classes que mantinguin les representacions de C++ en temps d'execució i operar amb elles normalment, però, a la vegada crear, per cada classe d'enumeració o de flags, una taula compartida que contingui les cadenes de text que representen els diferents símbols.

En el cas dels flags, el resultat no era suficientment pràctic donat que el codi d'usuari era molt redundat. La solució va ser implementar una interfície web en php que generava fitxers de definició de flags, i un script en el servidor de CVS que convertia les descripcions en codi.

Aquesta funció funcionava molt bé internament, però, un cop la llibreria fos de domini públic, la interfície web no era bona solució així que, finalment, la solució ha estat en fer aquest generador de flags executable externament al servidor CVS com a utilitat de la llibreria.

7.5.2 Contenidors

Amb alguns algorismes provinents de la versió antiga de SMS, hem heretat algunes estructures de dades genèriques. Arrays, Lists... El grup de desenvolupadors està molt dividit respecte de l'ús d'aquests contenidors en front dels que la STL ofereix.

En contra tenen el fet de que no són contenidors massa madurs i encaren apareixen inacabables errors de funcionament. Però, per un altre costat està el costum dels

usuaris a aquesta interfície, i el fet de que les estructures són, de fet components.

Alguns desenvolupadors hem fet templates per convertir qualsevol contenidor STL en un component i fins i tot versions de les estructures SMS que estan implementades internament amb STL.

En tot cas, tant si es fa servir els adaptador sobre la interfície STL com si fem servir els contenidors SMS, necessitem implementar la serialització en aquests objectes.

Fem servir la selecció estàtica de mètode per escollir quin tipus de serialització fem servir, de forma similar a com es fa als adaptadors `XMLContainerAdapter`.

Capítol 8

Conclusions

8.1 Resultats obtinguts

Com a resultat d'aquest projecte, s'ha aconseguit el principal objectiu d'integrar CLAM amb XML. Gairebé qualsevol objecte del model, té representació en XML, ja sigui com a component o com a tipus bàsic.

La feina requerida per l'usuari de CLAM per obtenir suport XML de nous objectes ha estat reduïda a mínims. La majoria d'objectes del model tenen una representació XML sense adoptar cap mesura de programació extra. Tenim suport automàtic d'entrada i sortida XML.

Tot i donar una implementació automàtica, si aquesta representació XML no és la que cerca l'usuari, pot redefinir-la fàcilment.

El suport automàtic abasta els objectes de configuració, els objectes de dades, i qualsevol altre objecte que sigui dynamic type. Per la resta (estat de processos, xarxes de processos, estructures de flux...), tot i que no tenen suport automàtic, el que cal fer per obtenir una implementació XML, és mínim i es redueix al primer nivell de composició.

En quant als objectes bàsics, hem aconseguit suport XML sense tenir que modificar la declaració de tipus. Calia evitar haver de modificar la declaració donat que molts tipus bàsics com els tipus nadius de C, no es poden modificar.

S'han aconseguit suplir molts dels mecanismes d'introspecció en temps d'execució que altres llenguatges fan servir per obtenir passivació automatitzada, per mecanismes estàtics en temps de compilació mitjançant la metaprogramació.

Els mecanismes estàtics, tot i ser menys flexibles que els mecanismes dinàmics, estan més optimitzats i els errors interns dels mecanismes d'automatització es detecten en temps de compilació. Als mecanismes dinàmics, la gestió dels tipus, pot estar mal gestionada i no adonar-nos fins que no ens dona un error en execució, quan, als mecanismes estàtics, es detecten en temps de compilació.

La pèrdua de flexibilitat seria greu si evités la incorporació de nous tipus externs a CLAM. No és tan greu donat que en inserir part del codi de gestió estàtica de tipus com a codi d'usuari generat automàticament, l'acoblament estàtic es fa en direcció de la part nova a la part CLAM.

En resum, el resultat és un sistema flexible i complert on es minimitzen l'esforç dels usuaris de la llibreria.

8.2 Aplicació i utilitat

Amb el suport XML que he proporcionat, és possible implementar totes les possibles aplicacions que s'explicaven a l'apartat 5.9. La majoria està funcionant ja, amb excepcions, donat que CLAM està encara desenvolupant-se i no estan implementats el control de flux automatitzat i la composició dinàmica de processos.

La seva utilització més brillant és com a interfície de configuració. Les aplicacions que fan servir CLAM es poden beneficiar del suport XML oferint als usuaris fitxers de configuració estructurats en XML. Per obtenir aquest suport, només cal crear un objecte de configuració amb els atributs que necessitem i fer-lo servir a l'aplicació

d'usuari per consultar o modificar aquest atributs amb els mateixos mètodes accessors com si fos un objecte qualsevol. Els atributs poden ser simples o poden ser altres objectes de configuració, o alguna dada complexa. Amb dos línies de codi podem tenir aquest objecte en disc i amb dos línies el podem recuperar.

Les dades de processament són altre grup d'objectes que es beneficia del suport XML dels dynamic types. Les aplicacions de poder serialitzar les dades de processament són múltiples i també es comenten a l'apartat 5.9: Els dynamic types faciliten la construcció de nous objectes mitjançant la composició. Així es poden aconseguir objectes complexos.

El llistat 8.1 és un exemple real d'utilització del sistema XML de CLAM en un projecte europeu anomenat CUIDADO que treballa en la extracció de descriptors en format MPEG-7 Audio, estàndard explicat a l'apartat 4.10. El llistat representa el resultat de l'extracció d'una melodia a partir d'un fragment d'àudio.

Com explicavem a l'apartat 5.9, la possibilitat de llegir i escriure dades en XML obre les portes no només a la activació i passivació de dades per a un sistema, sinó a la interacció entre sistemes mitjançant XML.

Per exemple, els anàlisis de melodia de CUIDADO s'estan fent servir com a entrada per a una altra aplicació anomenada SALTO, que sintetitza un saxòfon per models físics. SALTO abans es podia controlar només amb MIDI, i, amb el suport XML, ara també amb melodies de CUIDADO.

XML també ha resultat molt útil com a einta de desenvolupament facilitant la inspecció de dades en temps d'execució durant la depuració del programa. No cal oblidar que CLAM és una llibreria que faran servir altres desenvolupadors i aquest tipus d'ajudes al desenvolupament són d'agrair.

Com es comenta a l'apartat 5.6, en la primera fase de CLAM, no es contempla el mode supervisat. És per això que dos de les funcionalitats XML mencionades a l'apartat 5.9 no es poden fer servir encara.

Una de les funcionalitats que encara no són utilitzables és la de passivar i activar

```
<Melody>
    <noteArray>
        <Note>
            <pitchNote>
                <pitch>D</pitch>
                <octave>5</octave>
            </pitchNote>
            <fundFreq>594.063</fundFreq>
            <energy>0.0673407</energy>
            <time>
                <begin>0.161</begin>
                <end>3.326</end>
            </time>
        </Note>
        <Note>
            <pitchNote>
                <pitch>C</pitch>
                <octave>5</octave>
            </pitchNote>
            <fundFreq>525.408</fundFreq>
            <energy>0.0157708</energy>
            <time>
                <begin>3.462</begin>
                <end>6.697</end>
            </time>
        </Note>
    ...
        <Note>
            <pitchNote>
                <pitch>Bb</pitch>
                <octave>4</octave>
            </pitchNote>
            <fundFreq>473.401</fundFreq>
            <energy>0.0272985</energy>
            <time>
                <begin>7.473</begin>
                <end>7.768</end>
            </time>
        </Note>
    </noteArray>
</Melody>
```

Llistat 8.1: Descriptor de Melodia del projecte CUIDADO

xarxes de processos a disc. Tot i que, actualment, les xarxes de processos es fan servir, les conexions de flux estan programades directament. No es pot carregar una xarxa de processos arbitrària sense un control de flux automàtic que s'hi pugui adaptar en temps d'execució.

L'altra funcionalitat, que la falta de mode supervisat fa estèril, de moment, són les còpies de resguard de l'estat d'un sistema de processat. El problema és el mateix, tot i que els Processos són components i poden guardar el seu estat intern, encara no hi ha estructures de flux comunes a les que es pugui passivitzar.

8.3 Estudi econòmic

El projecte va començar al Febrer del 2001, al mateix temps que em vaig incorporar a l'equip que desenvolupa CLAM. Des de llavors he estat treballant un mínim de 32 hores setmanals, tot i que, evidentment no totes aquestes hores eren dedicades a quelcom relacionat amb projectes.

Al llarg d'aquest temps ha calgut realitzar tasques que no estan relacionades de forma directa amb aquest projecte: Suport a usuaris, manteniment dels sistemes, resolució d'incidències... També abans de poder fer alguns desenvolupaments centrats en el projecte ha calgut implementar algunes estructures comunes. Per això, el càlcul del temps de la taula 8.1 ha estat estimat respecte al temps total.

Activitat	Temps (hores)
Recerca	600
Disseny	400
Coordinació	180
Codificació	500
Documentació	200
Memòria	340
Total	2220

Taula 8.1: Distribució d'hores

Gran part de les hores dedicades han estat dedicades a recerca. A la taula 8.2 es fa

un detall per les àrea de recerca. El temps de coordinació representa el temps emprat en reunions de coordinació i fòrums on-line.

Àrea	Temps (hores)
C++	400
XML	150
Eines	50
Total recerca	600

Taula 8.2: Distribució d'hores de recerca

Com s'ha explicat al apartat 7.1.4, CLAM és un projecte molt documentat. Aquest temps inclou el de la documentació d'usuari, la documentació del codi i la documentació dels dissenys i propostes.

La codificació s'ha portat gairebé tantes hores com el disseny. Realment, la codificació, s'ha portat tant de temps per dos raons. Per un costat cal tenir en compte la magnitud de la llibreria i el temps que es triga en compilar-la cada cop que es fan canvis (és per això que fem prototipus, però tard o d'hora has d'integrar).

Per altre costat, cal tenir en compte el temps perdut fent diferents propostes al VisualC per que compilés sense que es pengés. En la taula 8.3, es pot veure la distribució d'hores segons les fases.

8.4 Línies futures

Un cop finalitzat aquest projecte, encara queda molta feina a fer en CLAM en general i en la part d'XML en concret. Al llarg de la memòria, ja s'han apuntat alguns aspectes que es podrien millorar o ampliar. A més, la mateixa implementació del sistema ha obert algunes portes que no semblaven obertes en la fase de disseny.

A continuació, s'enumeren les línies de futur, dins del mateix àmbit que aquest projecte de final de carrera, en les que es pot continuant fent recerca i desenvolupament.

La línia de futur que sembla més atractiva és reimplementar els procediments bàsics de

Àrea	Temps (hores)
Recerca prèvia	350
Sistema de composició	20
Gestió DOM de passivació	120
Adaptadors XML bàsics i array	20
Simbolització d'enums i flags	160
Passivació en els dynamic types	300
Estructures de dades	100
Gestió DOM per activació	320
Adaptadors per activació	100
Chained methods	430
Activació als dinàmic types	120
Adaptadors de contenidors iterables	10
Capceleres XML	40
Entrega	100

Taula 8.3: Distribució d'hores per fases

serialització considerant en el disseny les tècniques de metaprogramació. Les tècniques de metaprogramació es varen començar a introduir al projecte per resoldre problemes d'implementació un cop ja s'havia dissenyat i implementat parcialment el sistema. El coneixement d'aquestes tècniques ens podrien permetre un redisseny més net per fer el disseny més net. Per exemple, les interfícies estàtiques especificades de forma externa ens poden permetre no haver d'escollir el tipus d'adaptador enlloc.

La possibilitat d'implementar altres formats és una línia futura que ja estava prevista des del començament. En aquest sentit, ja hem fet les primeres reunions per adaptar del sistema de serialització per que suporti SDIF.

Caldria investigar si podem generar a la sortida XML el format final sense una representació DOM intermèdia. Amb objectes grans, aquesta representació intermèdia fa que la sortida XML no sigui pràctica. En ocupar tanta memòria el sistema esdevé lent.

A l'apartat 7.3.6, comento el problema que ens podem trobar si intentem escriure l'XML directament: Els atributs, malgrat l'usuari els insireixi després, s'han de treure abans a la sortida. A més, no basta amb controlar els atributs XML de l'element

visitat sinó que a més, cal controlar els atributs XML dels fills que són contingut pla que, com es veia a la figura 7.3 apareixen com atributs del element actual.

Tot i així, i donat el perjudici que crea aquesta representació intermèdia que en fa XercesC, valdria l'esforç investigar alternatives, per exemple, es podria fer que l'usuari retardi explícitament la sortida dels elements si es sap que després hi ha un atribut o un component com contingut pla.

Una altra situació en el que el sistema no és eficient és quan cal representar grans dades numèriques. La transcripció a decimal es porta molt temps i ocupa molt més espai. Al llarg del capítol 4 s'enfoca aquest problema des de diversos punts de vista. La solució més atractiva sembla la que proposa MPEG-7. Aquesta solució ens permet utilitzar l'actual sistema recursiu de serialització. Existeix una implementació disponible al public de l'algorisme de binarització i els mecanismes de selecció estàtica de mètode ens serien molt útils per determinar la codificació de les dades bàsiques.

El format binari necessita l'esquema, una bona funcionalitat a implementar podria ser la generació de l'esquema a partir de l'estructura estàtica.

Un altre tema, interessant és el suport de namespaces. De moment, tota la sortida XML està dintre d'un sol namespace. Quan els usuaris defineixin el seu propi namespace, per integrar XML generat o definit per dos entitats diferents.

Altre punt del sistema implementat que es podria millorar és la interfície amb el DOM. Actualment el maneig del DOM està encapsulat dintre del XMLStorage, alliberant la resta del sistema de dependències amb la llibreria. Però, la implementació interna del XMLStorage està fortament acoblada amb la llibreria XercesC. Posant un altre nivell d'abstracció entre el XMLStorable i la llibreria, oferint una interficie uniforme per la construcció i consulta del DOM, es podria aconseguir fer servir diverses llibreries com ara la de CenterPoint que sembla prou bona, la de Gnome o la de QT.

Per últim, mencionar una funcionalitat que també sembla interessant i implementable de forma directa: La capacitat d'afegir objectes a un document existent o llegir un objecte d'un punt concret del document. Amb la llibreria XalanC que implementa expressions XPath podriem situar-nos al punt DOM desitjat i llavors fer una activació

o passivació de l'objecte.

Bibliografia

[1] Agnula: A gnu/linux audio distribution. http://www.agnula.org.

[2] Blitz++, object-oriented scientific computing. http://www.oonumerics.org/blitz.

[3] Boost c++ libraries. http://www.boost.org.

[4] Boost preprocessor metaprogramming techniques. http://www.boost.org/libs/preprocessor/doc/tutorial.htm.

[5] Bscw, basic support for cooperative work. http://bscw.gmd.de/.

[6] Centerpoint/xml. http://www.cpointc.com/XML/.

[7] Free software foundation. http://www.gnu.org/fsf/fsf.html.

[8] Gdome, the gnome dom engine. http://www.levien.com/gnome/gdome.html.

[9] Gnu general public license. http://www.gnu.org/licenses/gpl.html.

[10] Microsoft xml sdk. http://msdn.microsoft.com/xml.

[11] Oracle xml developer's kits. http://www.oracle.com/xml.

[12] Portability hints: Microsoft visual c++ 6.0 sp4. http://www.boost.org/more/microsoft_vcpp.html.

[13] Spirit parser home. http://spirit.sourceforge.net.

[14] W3 consortium home page. http://www.w3.org.

[15] Xerces c++. http://xml.apache.org/xerces-c.

[16] The xml c library for gnome. http://xmlsoft.org.

[17] Mpeg-7 ddl working draft 4.0. http://archive.dstc.edu.au/mpeg7-ddl.

[18] Xml fragment interchange, w3c candidate recommendation, 12 Febrary 2001. http://www.w3.org/TR/xml-fragment.

[19] Xml linking language (xlink) version 1.0, w3c recommendation, 27 June 2001. http://www.w3.org/TR/xlink.

[20] Xml inclusions (xinclude) version 1.0 w3c candidate recommendation, 21 February 2002. http://www.w3.org/TR/xinclude.

[21] Amatriain, X. Metrix: A musical description language for a spectral modeling based synthesizer, 1999. http://www.iua.upf.es/~xamat/metrix.

[22] Arnold, S., Boehm, C., and Hall, C. Mutated. http://www.pads.ahds.ac.uk/mutated.

[23] Attardi, G., and Cisternino, A. Reflection support by means of template metaprogramming. In *GCSE* (2001), pp. 118–127.

[24] Ayars, J., Bulterman, D., Cohen, A., Day, K., Hodge, E., Hoschka, P., Hyche, E., Jourdan, M., Kim, M., Kubota, K., Lanphier, R., Layaïda, N., Michel, T., Newman, D., van Ossenbruggen, J., Rutledge, L., Saccocio, B., Schmitz, P., and ten Kate, W. Synchronized multimedia integration language (smil 2.0). http://www.w3.org/TR/smil20.

[25] Bassetti, F., Davis, K., and Quinlan, D. C++ expression templates performance issues in scientific computing. pp. 635–639.

[26] Batlle, E., and Cano, P. Automatic segmentation for music classification using competitive hidden markov models. In *International Symposium on Music Information Retrieval* (2000).

[27] Baus, C., and Becker, T. Custom iterators for the STL. In *First Workshop on C++ Template Programming, Erfurt, Germany* (October 10 2000).

[28] Beck, K. *Smalltalk Best Practice Patterns*. Prentice Hall, October 1996.

[29] Berglund, A., Boag, S., Chamberlin, D., Fernandez, M. F., Kay, M., Robie, J., and Siméon, J. Xml path language (xpath) 2.0, w3c working draft. http://www.w3.org/TR/xpath20.

[30] Berti, G. Generic components for grid data structures and algorithms with C++. In *First Workshop on C++ Template Programming, Erfurt, Germany* (October 10 2000).

[31] Bray, T., Paoli, J., Sperberg, C. M., and Maler, E. Xml 1.0 w3c recommendation. http://www.w3.org/XML.

[32] Breymann, U., Czarnecki, K., and Eisenecker, U. Generative components: One step beyond generic programming. http://nero.prakinf.tu-ilmenau.de/ czarn/dagstuhl.ps.

[33] Buhr, P. A., and Mok, W. Y. R. Advanced exception handling mechanisms.

[34] Castan, G. Musixml. http://www.s-line.de/homepages/gerd_castan/ compmus/MusiXML_e.html.

[35] Chang, B., Heninger, A., Kesselman, J., and Rahman, R. Document object model (dom) level 3 abstract schemas and load and save specification. http://www.w3.org/TR/2001/WD-DOM-Level-3-ASLS-20010607.

[36] Clark, J. Xsl transformations (xslt). http://www.w3.org/TR/xslt11.

[37] Cowan, J., and Tobin, R. Xml information set (infoset) w3c recommendation. http://www.w3.org/TR/xml-infoset/.

[38] Cox, B. J., and Novobilski, A. J. *Object-Oriented Programming An Evolutionary Approach*, second ed. Addison-Wesley, 1991.

[39] Czarnecki, K., Eisenecker, U., and Steyaert, P. Beyond objects: Generative programming.

[40] Day, N., and Martínez, J. M. Introduction to mpeg-7. http://www.darmstadt.gmd.de/mobile/MPEG7.

[41] de Boer, M., Bonada, J., and Serra, X. Using the sound description interchange format within the sms applications. In *Proceedings of International Computer Music Conference 2000* (2000).

[42] Dodds, L. Investigating the infoset.

[43] Eichelberger, H., and v. Gudenberg, J. W. UML description of the STL. In *First Workshop on C++ Template Programming, Erfurt, Germany* (October 10 2000).

[44] Eisenecker, U. W., Blinn, F., and Czarnecki, K. A solution to the constructor-problem of mixin-based programming in C++. In *First Workshop on C++ Template Programming, Erfurt, Germany* (October 10 2000).

[45] Fallside, D. C. Xml schema w3c recommendation. http://www.w3.org/XML/Schema.

[46] Fröhlich, A. A., and Schröder-Preikschat, W. Scenario adapters: Efficiently adapting components.

[47] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns-Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[48] Garzón, D. G., and Amatriain, X. Xml as a means of control for audio processing, synthesis and analysis. In *MOSART, Workshop on Current Research Directions in Computer Music* (Barcelona, November 2001). http://www.iua.upf.es/mtg/mosart/papers/p27.pdf.

[49] Gil, J. Y., and Gutterman, Z. Compile time symbolic derivation with C++ templates. pp. 249–262.

[50] Goldberg, A., and Robson, D. *Smalltalk-80: The Language and Its Implementation*. Addison-Wesley, 1983.

[51] Good, M. Musicxml. http://www.musicxml.org.

[52] Heuer, J., Thienot, C., and Wollborn, M. Binary format. In *Introduction to MPEG-7, multimedia content description interface*, B. S. Manjunath, P. Salembier, and T. Sikora, Eds. 2002.

[53] Highley, T., Lack, M., and Myers, P. Aspect oriented programming: A critical analysis of a new programming paradigm. Tech. Rep. CS-99-29, 5, 1999.

[54] Hors, A. L., Hégaret, P. L., Nicol, G., Wood, L., Champion, M., and Byrne, S. Document object model (dom) level 3 core specification. http://www.w3.org/TR/2001/WD-DOM-Level-3-Core-20010913.

[55] Hors, A. L., Hégaret, P. L., Nicol, G., Wood, L., Champion, M., Byrne, S., and Robie, J. Document object model (dom) level 2 core specification. http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113.

[56] Hors, A. L., Sultor, R., Wilson, C., Isaacs, S., Jacobs, I., Nicol, G., Wood, L., Champion, M., Byrne, S., and Robie, J. Document object model (dom) level 1 specification.

[57] ISO/IEC. Standard music description language (smdl). http://www.ornl.gov/sgml/SC34.

[58] Johnson, R. E. Dynamic object model. Creació de tipus en temps d'execució.

[59] Kaye, R., and Pouwelse, J. Musicbrainz metadata initiative. http://www.musicbrainz.org/MM.

[60] Kreft, K., and Langer, A. Combining oo design and generic programming. *C++ Report* (March 1997).

[61] Kühl, D. STL and OO don't easily mix. In *First Workshop on C++ Template Programming, Erfurt, Germany* (October 10 2000).

[62] Manjunath, B. S., Salembier, P., and Sikora, T., Eds. *Introduction to MPEG-7, multimedia content description interface*. Wiley, 2002.

[63] McNamara, B., and Smaragdakis, Y. Static interfaces in C++. In *First Workshop on C++ Template Programming, Erfurt, Germany* (October 10 2000).

[64] Megginson, D. Sax 2.0, simple api for xml. http://sax.sourceforge.net.

[65] Pixley, T. Document object model (dom) level 3 events specification 1.0 w3c working draft. http://www.w3.org/TR/DOM-Level-3-Events.

[66] Reis, G. D. Expressing constraints a la sml. In *ACCU Spring Conference* (2001).

[67] Reis, G. D. Metaprogramming in c++. In *ACCU Spring Conference* (2001).

[68] Schiettecatte, B. A format for virtual orchestras: Flowml. http://wendy.vub.ac.be/~bschiett/saol/FlowML.html.

[69] Serra, X., and Smith, J. Spectral modeling synthesis. In *Proceedings of International Computer Music Conference 1989* (Ohio, USA, 1989).

[70] Siek, J. A modern framework for portable high performance numerical linear algebra. http://lsc.nd.edu/downloads/research/mtl/papers/thesis.pdf, 1999.

[71] Siek, J., and Lumsdaine, A. Concept checking: Binding parametric polymorphism in C++. In *First Workshop on C++ Template Programming, Erfurt, Germany* (October 10 2000).

[72] Steyn, J. Music markup language. http://www.mmlxml.org.

[73] Striegnitz, J., and Smith, S. A. An expression template aware lambda function. In *First Workshop on C++ Template Programming, Erfurt, Germany* (October 10 2000).

[74] Stroustrup, B. *The C++ Programming Language*, second ed. Addison Wesley, 1991.

[75] Stroustrup, B. *The Design and Evolution of C++*. Addison-Wesley, 1994.

[76] Stroustrup, B. Why c++ is not only an object-oriented programming language. In *OOPSLA'95 Proceedings* (1995).

[77] van Rotterdam, J. Musicml, an xml experience. http://www.tcf.nl/3.0/musicml.

[78] Veldhuizen, T. L. C++ templates as partial evaluation. In *Partial Evaluation and Semantic-Based Program Manipulation* (1999), pp. 13–18.

[79] Veldhuizen, T. L. Five compilation models for C++ templates. In *First Workshop on C++ Template Programming, Erfurt, Germany* (October 10 2000).

[80] Vlissides, J. Pattern hatching: Visiting rights. *C++ Report* (September 1995).

[81] Vlissides, J. Pattern hatching: Visitor in frameworks. *C++ Report* (November/December 1999).

[82] Weiser, M., and Powell, G. The View Template Library. In *First Workshop on C++ Template Programming, Erfurt, Germany* (October 10 2000).

[83] Whitmer, R. Document object model (dom) level 3 xpath specification 1.0 w3c working draft. http://www.w3.org/TR/DOM-Level-3-XPath.

[84] Wright, M., Chaudhary, A., Freed, A., Wessel, D., Rodet, X., Virolle, D., Woehrmann, R., , and Serra, X. New applications of the sound description interchange format. In *Proceedings of the International Computer Music Conference* (1998).

Índex alfabètic