Université Joseph Fourier
U.F.R
Informatique et Mathématiques Appliquées

I.M.A.G.

MAGISTERE D'INFORMATIQUE

Projet présenté par :

Lionel VILLARD

Mise en œuvre d'un environnement multimédia

Effectué dans le projet Opéra − INRIA

Date : Septembre 1998
Jury : Cécile Roisin
Claude Puech

Mise en œuvre d'un environnement multimédia

Mise en œuvre d'un environnement multimédia

Lionel Villard

INRIA Rhône−Alpes − Projet OPERA

1 Introduction

1.1 Contexte

Le projet OPERA a pour thème général les applications de traitement des documents électroniques : modélisation des documents et réalisation d'environnements d'édition interactive et coopérative de documents structurés multimédia. Les travaux de modélisation ont permis notamment de caractériser les documents selon quatre dimensions : logique, spatiale, hypermédia et temporelle. Des outils d'édition s'appuient sur ces quatre modes d'organisation :

- Thot [Qui87], un système général et paramètrable pour l'édition interactive de documents conventionnels structurés ;
- Alliance [Dec96] et Byzance, des systèmes d'édition coopérative ;
- Madeus [Lay97], un prototype pour l'édition et la présentation de documents multimédia.

Ce dernier est au cœur de ce rapport. Avec cet outil, le projet OPERA vise à offrir un environnement d'édition de document multimédia dont un des objectifs est de fournir un langage d'expression riche et simple pour l'ordonnancement temporel et le placement spatial.

Pour l'ordonnancement temporel, l'approche choisie est basée sur la spécification déclarative de contraintes entre les objets. Ces contraintes expriment le placement temporel relatif des objets (les uns par rapport aux autres). Par exemple, un auteur voulant exprimer le scénario : « *La vidéo A se joue avant la vidéo B, qui elle même se jouera en même temps que la vidéo C* », le traduira par les deux contraintes suivantes :

- *Vidéo A* **avant** *Vidéo B*
- *Vidéo B* **égale** *Vidéo C*

En ce qui concerne le placement spatial, deux approches ont été choisies, le placement direct et le placement relatif. Ce dernier s'apparente à l'ordonnancement temporel. Par exemple, on peut spécifier que les objets *Vidéo A* et *Vidéo B* sont alignés sur leur bord

supérieur et que l'objet *Vidéo A* est à droite de l'objet *Vidéo B*. Cela ce traduit par les deux contraintes suivantes :

- *Vidéo A* **aligné_bord_supérieur** *Vidéo B*
- *Vidéo A* **à_droite_de** *Vidéo B*

Les concepteurs de Madeus ont considéré dans un premier temps ces deux dimensions comme étant indépendantes, ceci au niveau de la spécification et, par conséquent, au niveau de l'exécution. Le couplage de ces dimensions, effectué durant mon stage de DEA, a abouti vers une spécification d'attributs et de relations spatio−temporelles.

Depuis avril 1998, Madeus subit une refonte profonde tant au niveau de architecture que son implantation. La nouvelle version de ce logiciel, appelé Madeus 2.0, repose maintenant sur une hiérarchie de classes implémentées en utilisant le langage Java. L'objectif est d'avoir un environnement multimédia qui soit portable et facilement extensible grâce d'une part à l'approche orienté objet, et d'autre part aux nombreuses classes Java existantes (Graphiques, JMF, etc.). Un autre objectif est d'offrir un système multimédia plus complet au niveau de l'édition par rapport à Madeus 1.0 [Jou98a][Jou98b].

Le travail effectué dans le cadre de ce magistère est une contribution à cette nouvelle version de Madeus. Ainsi, avant de présenter les objectifs de ce travail, nous décrivons l'architecture globale de Madeus 2.0.

1.2 Architecture globale de Madeus 2.0

Madeus 2.0 repose sur les concepts de *document* et de *vue*. Le document représente la partie logique d'un document multimédia, tandis que la vue correspond à une représentation physique particulière. La gestion des documents est répartie principalement en trois classes (c.f. figure 1) :

- la classe **MadeusManager** gère, entre autres, l'ensemble des documents ouverts. Ainsi, le concept de multi−documents est introduit dans Madeus. Ce concept permet notamment la navigation entre plusieurs documents (hyperlien inter−document). De plus, cette classe contient le gestionnaire temporel et spatial, implanté respectivement dans les classes **TemporalManager** et **SpatialManager**.
- la classe **DocumentManager** gère le contexte nécessaire à un document. Il contient notamment un gestionnaire multi−vues (*ViewManager*) ;
- la classe **Parser** permet d'interpréter un code source pour produire un document géré par la classe *DocumentManager*.

A chaque document sont associées une ou plusieurs vues (relation 0..*). La notion de multi−vues est gérée dans la classe *ViewManager*. Cette classe gère un ensemble de

vues de type différent. Par exemple, la vue **TextualView** permet de visualiser sous la forme textuelle le document (c'est−à−dire voir et éditer le code source), tandis que la vue **ExecutionView** permet de jouer le document.

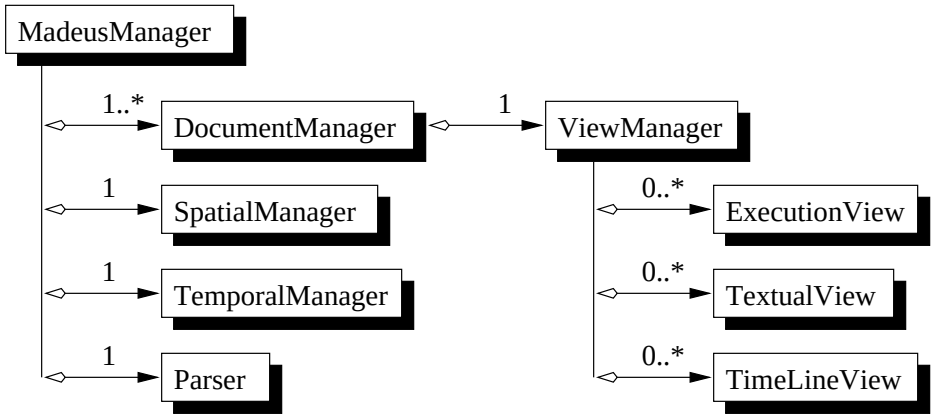

Figure 1 : Architecture de Madeus 2.0

1.3 Objectifs de mon travail de magistère

Dans ce cadre applicatif, les objectifs de ce magistère sont doubles. Il s'agit dans un premier temps de mettre en œuvre dans Madeus 2.0 la spécification des relations spatio−temporelles telle qu'elle a été définie dans le cadre de mon projet de DEA. Ensuite, dans un second temps, l'objectif est d'apporter une solution théorique au problème évoqué dans mon rapport de DEA sur la vérification des relations spatio−temporelles.

Pour mener à bien ces objectifs, j'ai participé aux spécifications et au développement de parties communes de l'application Madeus 2.0 dans la mesure où le début de son développement a coïncidé au début de ce stage. Ce travail s'inscrit donc dans le cadre d'un projet commun d'une équipe de recherche impliquant cinq personnes de l'équipe OPERA.

La suite de ce rapport est organisé comme suit :

- dans la section 2 nous proposons, après un bref rappel de la spécification des attributs et des relations spatio−temporelles, une DTD(1) XML(2) afin d'exprimer ce type de relations. Elle décrit ensuite les choix et la mise en œuvre du parser associé (bloc *Parser* de la figure 1*)* ;
- dans la section 3 nous décrivons les différentes étapes pour créer la vue d'exécution afin de jouer des documents intégrant la spécification spatio−temporelle (bloc *ExecutionView* de la figure 1*)* ;

- le problème de la vérification spatio−temporelle est étudié dans la section 4. Les points de vérification à effectuer sont identifiés, et nous verrons ce que le système de vérification actuel est capable de vérifier ;
- enfin, dans la section 5 nous proposons les perspectives envisagées à la suite de ces travaux, puis termine par le bilan de ce stage.

2 Modèle de document Madeus

Le besoin d'écrire un document Madeus nous a mené vers l'étude d'un choix de syntaxe. Ce choix s'est orienté vers le langage de marquage XML pour plusieurs raisons. XML est un standard reconnu, simple de lecture et d'écriture, et enfin, il existe plusieurs implémentations en Java de parser XML.

Dans cette section, après un rappel de la spécification de relations et des attributs spatio−temporels définis dans mon DEA, nous présentons brièvement le standard XML. Ensuite, nous présentons le format en XML de nos documents multimédia intégrant l'expression spatio−temporelle.

2.1 Documents Madeus avec l'expression spatio−temporelle

Durant mon projet de DEA, j'ai proposé un modèle pour spécifier dans un premier temps les attributs spatio−temporels, puis des relations spatio−temporelles. Dans cette section, nous présentons brièvement les principaux concepts de cette spécification.

2.1.1 Attributs spatio−temporels

Informellement, un attribut spatio−temporel est un attribut de présentation (position, couleur, etc.) défini par une ou plusieurs fonctions dépendantes du temps. Chaque fonction est associée à un *sous−intervalle spatio−temporel.* La figure 2 illustre ces notions par un effet de fondu : à l'état initial, la couleur de l'objet est transparente, puis devient progressivement noire en 2 secondes, reste noire pendant une seconde, puis redevient progressivement noire.

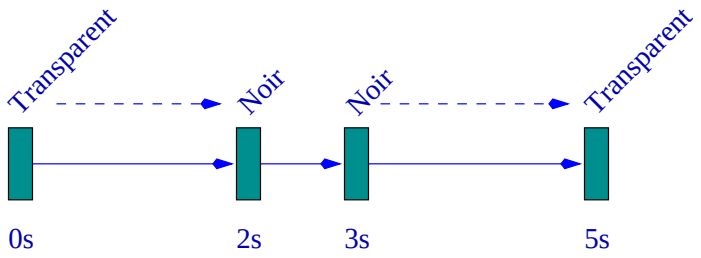

Figure 2 : Un effet de fondu

2.1.2 Relations spatio−temporelles

Deux catégories de relations spatio−temporelles existent. La première catégorie contient les relations spatiales qui dépendent du temps. Elles permettent de définir la valeur d'un attribut spatio−temporelle. La deuxième catégorie des relations spatio−temporelles contient les relations temporelles dépendantes du spatial.

2.1.2.1 Relations spatiales en fonction du temps

Ce sont des relations spatiales ayant une durée de vie. On distingue trois types de relations spatiales en fonction de leur durée de vie :

- **Relation spatiale intemporelle**

  La relation spatiale est valide pendant toute le durée de vie d'au moins un des deux objets en cause.

- **Relation spatiale ponctuelle**

  La relation spatiale est valide à un instant donné. Cet instant est défini, par convention, comme étant l'instant final d'un sous−intervalle. Les relations spatiales de Madeus 1.0 sont de ce type.

- **Relation spatiale temporisée**

  La relation spatiale est valide durant un intervalle de temps non nul. La durée de validité est soit spécifiée directement au niveau de la relation, soit via l'attribut *durée* d'un objet qui peut être un autre que les deux objets en relation. Par exemple, l'objet *A* peut être aligné à gauche avec l'objet *B* pendant la durée de l'objet *C* (c.f. figure 3).

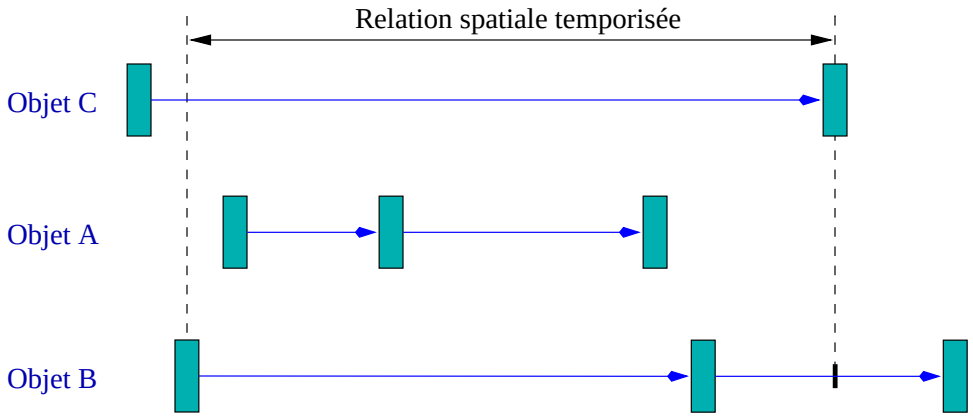

Figure 3 : Une relation spatiale temporisée

2.1.2.2 Relations temporelles en fonction du spatial

Du fait que les objets sont maintenant dynamiques, les possiblités de synchronisation augmentent en tenant compte de la valeur des attributs spatiaux du document. Par exemple, l'auteur peut spécifier qu'un objet A commence lorsqu'un autre objet B est à la position (10, 10).

Dans la plupart des cas, la synchronisation spatiale peut se réduire à une synchronisation temporelle. En effet, sur l'exemple précédent, si l'instant auquel l'objet B arrive à la position (10, 10) peut être calculé, alors il suffit de remplacer la synchronisation spatiale par la synchronisation temporelle. Par contre, si l'instant ne peut pas être calculé, alors la relation est de caractère événementielle. Nous ne traiterons pas de ce dernier aspect dans ce rapport puisqu'il est relativement difficile de l'intégrer dans Madeus puisque celui−ci repose sur une approche relationnel.

2.2 Présentation de XML

XML est un standard défini par un groupe de travail du W3C[(3)]. XML est très proche de son prédécesseur SGML[(4)][ISO86]. Il répond aux mêmes objectifs, c'est−à−dire décrire selon un format déclaratif la structure arborescente d'un document. Il repose aussi sur le concept de DTD permettant de définir une classe de document. Chaque document correspond alors à une instanciation de la DTD.

Le standard XML contient deux jeux de syntaxe, l'un pour créer une DTD et l'autre pour écrire une instance de document. Nous décrivons ci−dessous les principaux traits du langage de marquage, à savoir les définitions de types d'éléments et d'attributs.

2.2.1 Les éléments

La définition d'un élément dans la DTD s'effectue de la façon suivante :

```
<!ELEMENT NomElement ModèleDeContenu>
```

Quatre types de modèle de contenu sont distingués :

- le mot clé **EMPTY** indique que l'élément est vide, c'est−à−dire qu'il n'a pas de contenu. L'instance dans ce cas à la forme suivante :

```
<NomElement/>
```

- le contenu peut être constitué d'autres éléments. La description de ce modèle de contenu reprend la notation utilisée pour décrire une expression régulière (c.f [W3C98b]). L'exemple de la figure 4 définit un élément date contenant trois éléments successifs, le jour, le mois et l'année ;

```
DTD :
      <!ELEMENT date (jour, mois, année)>
      <!ELEMENT jour (#PCDATA)>
      <!ELEMENT mois (#PCDATA)>
      <!ELEMENT année (#PCDATA)>

Exemple d'instance :
      <date>
          <jour>20</jour>
          <mois>08</mois>
          <année>1998</année>
      </date>
```

Figure 4 : Exemple de contenu

- le contenu est un mélange entre des éléments et des chaînes de caractère (noté *#PCDATA*). Leur composition s'effectue uniquement via l'opérateur de choix (« | ») et le groupe de contenu est obligatoirement suivi de l'opérateur d'itération « * » (c.f exemple figure 5) ;

```
DTD :
      <!ELEMENT txtdate
                (#PDATA | jour | mois)*>


Exemple d'instance :
      <txtdate>
          On est le <jour>2</jour>
          <mois> septembre </mois> 1998
      </txtdate>
```

Figure 5 : Mélange entre des éléments et des chaînes de caractères

- enfin, le mot clé **ANY** indique que l'élément peut contenir n'importe quel type de contenu.

2.2.2 Les attributs

A chaque élément peut être associée une liste d'attributs (exemple figure 6). La définition de cette liste s'effectue comme suit :

```
<!ATTLIST NomElement
          NomAttribut TypeAttribut ValeurParDefault
          ...>
```

NomElement correspond à un élément défini. L'attribut est soit de type chaîne de caractères (notée *CDATA*), de type énumération, ou de type mot−clé. Par exemple le mot−clé *ID* signifie que la valeur de l'attribut est unique pour l'élément contenant cet attribut.

Enfin il existe trois types de valeur par défaut :

- **#REQUIRED**

  l'instance d'un élément doit obligatoirement fournir une valeur pour cet attribut, sinon le document n'est pas valide ;

- **#IMPLIED**

  l'attribut est facultatif ;

- **#FIXED** *ValeurParDefaut*

  si l'attribut a une valeur dans l'instance, alors elle doit être identique à celle spécifiée dans la DTD.

```
DTD:
      <ELEMENT date EMPTY>
      <ATTLIST date
               clé   ID      #REQUIRED
               jour  CDATA   #REQUIRED
               mois  CDATA   #REQUIRED
               année CDATA   #IMPLIED>
Exemple d'instance:
      <date clé   = "Mon anniversaire"
            jour  = "19"
            mois  = "Novembre">
```

Figure 6 : Exemple de définition d'attributs avec une instance

2.3 DTD Madeus

La version précédente de Madeus reposait sur un langage de marquage ad−hoc, auquel l'expression spatio−temporelle a été rajouté. Nos objectifs sont d'une part de définir un nouveau langage qui intègre de façon homogène les traits du langage précédent avec l'expression spatio−temporelle, et d'autre part d'offrir un langage évolutif, notamment sur les objets de base des documents.

La DTD que nous proposons repose sur les constructeurs de base, les éléments et les attributs, décrits dans la section précédente. La difficulté de la définition d'une DTD réside dans le choix entre ce qui est un élément et ce qui est un attribut. Ce choix a été guidé par les principes suivants :

- les informations portées par les attributs sont celles qui n'ont pas besoin d'être décomposées (informations terminales) ;
- les informations de nature structurale sont déclarées sous la forme d'éléments.

Dans la suite de cette section, nous présentons quelques extraits simplifiés (pas d'entités) de la DTD, extraits surtout orientés vers la spécification spatio−temporelle. L'Annexe A contient la DTD complète du langage de Madeus.

2.3.1 Notation

Pour décrire notre DTD, nous utilisons les notations suivantes :

- mots en majuscule : mot−clé de la grammaire décrivant le langage de la DTD, par exemple le mot−clé ELEMENT ;
- mots en minuscule : mot−clé de la grammaire décrivant l'instance de la DTD, comme le mot−clé madeus.

2.3.2 L'élément *madeus*

```
<!ELEMENT madeus
          (text | video | audio | picture |
           external | composite)>
<!ATTLIST madeus
           name        CDATA #REQUIRED
           background  CDATA #IMPLIED>
```

Un document Madeus est caractérisé par un ensemble d'attributs comme son nom et son fond d'écran. Son contenu est soit un objet de base (text, image, video, etc.), soit un objet composite. L'objet particulier *external* inclut tous les objets de base non reconnus, dont la mise en œuvre respecte une certaine interface afin de pouvoir les manipuler. Cela permet d'ajouter de nouveaux objets avec un minimum de modification. Le principe de fonctionnement est identique à celui des *plugins.*

2.3.3 Les objets de base

Pour chaque type d'objet de base, un type d'élément avec les attributs correspondants est défini. Par exemple, pour le type de base *text* on a :

```
<!ELEMENT text
          (text_functions, text_intervals?,
           relations?)>
<!ATTLIST text
          name      CDATA #REQUIRED
          duration  CDATA #IMPLIED
          left      CDATA #IMPLIED
          top       CDATA #IMPLIED
          font_name CDATA #IMPLIED>
```

Un objet de base est caractérisé par un ensemble de propriétés communes à tous les objets comme le nom de l'objet (*name*) qui doit être unique au sein du composite. Ces propriétés incluent aussi les propriétés spécifiques au type de l'objet, comme la fonte lorsque l'objet est de type texte, ou encore le volume pour un objet de type audio. La liste exhaustive par type d'objet des propriétés est donnée dans [Vil98]. Ces propriétés sont spécifiées grâce aux attributs XML. Les propriétés définies à ce niveau sont affectées pour toute la durée de l'objet. Ce sont les propriétés par défaut.

Le contenu de cet élément contient une liste de fonctions (c.f [Vil98]), puis une liste des sous−intervalles de l'objet, suivi des relations entre ces intervalles. Les fonctions s'appliquent sur les propriétés. Par exemple, la fonction *Move*, qui s'applique sur la propriété *Position*, permet de déplacer un objet visuel (texte, image, vidéo). Cette notion de sous−intervalle qui n'existait pas dans le prototype Madeus 1.0, nous permet de mettre en œuvre les propriétés et les relations spatio−temporelles. Nous pouvons remarquer aussi

qu'il est facultatif de spécifier des sous−intervalles (point d'interrogation de l'élément *text_intervals*), ce qui correspond au format pivot de Madeus 1.0.

Un intervalle pour un objet de type *text* est de la forme suivante :

```
<!ELEMENT text_intervals (text_interval+)>
<!ELEMENT text_interval   text_functions>
<!ELEMENT text_functions (move | rotate | scale)*>
<!ATTLIST text_interval
          name     CDATA #REQUIRED
          duration CDATA #IMPLIED
>
```

Pour chaque objet, il est donc possible d'associer un ensemble de sous−intervalles spécifiant, de la même façon que pour l'objet, les valeurs associées aux propriétés ainsi que des fonctions. La spécification des propriétés dans un sous−intervalle surcharge la spécification au plus haut niveau (objet et composites). Ce système d'héritage est voisin de celui de CSS2 (c.f. [W3C98a]).

2.3.4 L'objet composite

Un objet composite est défini de la façon suivante :

```
<!ELEMENT composite (all_properties*,
                     (text | audio | video | image
                      | external | composite)+,
                      relations)>
<!ATTLIST composite
          name                  CDATA #REQUIRED
          composite_properties>
```

Un composite est caractérisé par un ensemble de propriétés (*nom*, *position*, etc.). Le contenu inclut les propriétés par défaut pour tous les descendants du composite. Comme un composite peut contenir tout type d'objet, il peut définir une valeur par défaut sur l'union de toutes les propriétés des objets.

Ensuite les fils du composite sont spécifiés : ce sont soit des objets de base, soit des composites. Enfin, les relations entre les intervalles déclarés dans le composite sont spécifiées. Un intervalle est soit l'intervalle de l'objet, soit un sous−intervalle. Dans ce cas, le nom du sous−intervalle est préfixé par le nom de l'objet.

2.3.5 Les relations

La liste des relations intra et inter−propriétés est spécifiée selon une forme qui dépend de la nature de la relation. Par exemple, la relation d'alignement à gauche est décrite par :

```
<!ELEMENT left_spacing EMPTY>
<!ATTLIST left_spacing
          delay           CDATA #IMPLIED
          first_interval  CDATA #REQUIRED
          second_interval CDATA #REQUIRED>
```

Le nom de l'élément correspond au nom de la relation, dans ce cas *left_spacing*, et les attributs de la relation sont les opérandes de celle−ci. Les intervalles mis en relation appartiennent obligatoirement au composite dans lequel la relation est définie.

Parfois la relation a besoin de plusieurs paramètres. En fonction de la relation, des attributs sont alors ajoutés sous la forme d'attributs XML.

2.3.6 Exemple

Cet exemple (c.f. figure 7) est constitué d'un composite dans lequel deux objets sont définis : un objet de type texte et un autre de type image. Initialement, l'objet de type texte est à la position (10, 40). Lorsque l'objet *Ballon* de type image commence à se déplacer de la position (50, 250) vers la position (100, 40), le sous−intervalle *i2* de l'objet *Barthez* est activé : l'objet de type texte se déplace pour terminer sa trajectoire de façon à être centré verticalement avec *Ballon*.

```
<?xml version = "1.0"?>
<!DOCTYPE madeus SYSTEM "madeus.dtd">

<madeus name = "Coupe du Monde"
        background = "gazon.jpg">

<composite name = "Arrêt spectaculaire">

    <image name = "ballon"
           value = "ballon.bmp"
           duration = "10s">
       <move src_x = "50" src_y = "250"
             dst_x = "100" dst_y = "40">
    </image>

    <text name = "barthez"
          value = "Mur infranchissable">
        <intervals>
            <interval name = "i1"
                      Left = "10" Top = "40">
            </interval>
            <interval name = "i2">
            </inteval>
        </intervals>

    </text>

    <relations>
      <starts      interval_one = "ballon"
                   interval_two = "barthez.i2"/>
      <center_vert duration = "0"
                   interval_one = "ballon"
                   interval_two = "barthez.i2"/>
    </relations>
</composite>

</Madeus>
```

Figure 7 : Exemple d'une instance d'un document Madeus en XML

2.3.7 Synthèse

Cette syntaxe permet de spécifier des documents multimédia telle qu'ils ont été définis dans [Lay97]. De plus, elle permet de spécifier des attributs et des relations spatio−temporelles telles qu'ils ont été définis dans [Vil98].

Ce format ne sépare pas la structure logique du document de sa présentation, ceci par un souci de simplicité. D'autres approches de traitement de documents l'ont fait (Thot, application SGML, etc.).

2.4 Mise en œuvre du parser

Comme il existe relativement beaucoup d'implantation de parser XML en Java, nous avons décidé, par soucis de gain de temps, d'utiliser une implantation existante. Lors de nos investigations, nos critères de sélection ont été les suivants :

- le parser doit être libre d'utilisation, et de distribution ;
- il doit être écrit en « pur Java » afin d'être portable ;
- il doit être validant, c'est−à−dire capable d'interpréter une DTD et de vérifier que l'instance respecte cette DTD ;
- il doit respecter le standard XML 1.0 tel qu'il a été défini par le W3C ;
- il doit aussi respecter un des standards d'implantation d'un parser XML, à savoir le standard DOM (Document Object Model [W3C98c], ou le standard SAX (Simple API for XML [Tig97]). Cela permettra d'utiliser plusieurs implantations de parser et répondre ainsi à une évolutivité au niveau du standard XML ;
- et enfin, il doit être stable.

Le seul parser répondant à tous ces critères est *xml4j*, le parser d'IBM. Il implante le standard SAX qui repose sur un modèle par événements. Il suffit donc d'implanter une interface réagissant aux événements produits par le parser. Ces événements sont notamment l'ouverture du document, l'ouverture d'un élément, et la fermeture d'un élément.

L'exécution du parser produit une instance d'une classe appelée « document de référence » utilisé par les classes gestionnaires des vues de l'application Madeus.

3 Système d'exécution

Un document multimédia est joué grâce au système d'exécution. Dans Madeus 2.0, le système d'exécution est réalisé à travers une vue particulière du document appelé vue d'exécution (c.f figure 1). Dans cette section, nous commençons par présenter l'architecture globale de cette vue d'exécution. Puis nous détaillons les deux composantes principales d'une vue, c'est−à−dire le document et la fenêtre. Nous supposons que le gestionnaire temporel existe (c.f. [Lay97]), nous nous intéressons donc uniquement au gestionnaire spatial.

3.1 Architecture de la vue d'exécution

La vue d'exécution est constituée d'une classe **ExecutionView** qui est chargée de créer une instance de la classe **ExecutionDocument** à partir de l'instance de la classe **Document** en interprétant les attributs spécifiques à cette vue. Elle crée aussi une

instance de la classe **ExecutionWindow** permettant de visualiser un tel document. Les communications entre le document exécutable et la fenêtre d'exécution s'effectuent via la classe *ExecutionView*.

Cette architecture respecte les modèles d'architecture conçus dans le domaine de l'interface homme−machine, comme PAC [Cou87] ou MVC. Son intérêt principal est de séparer les données de leur représentation afin d'offrir une plus grande souplesse d'évolution.

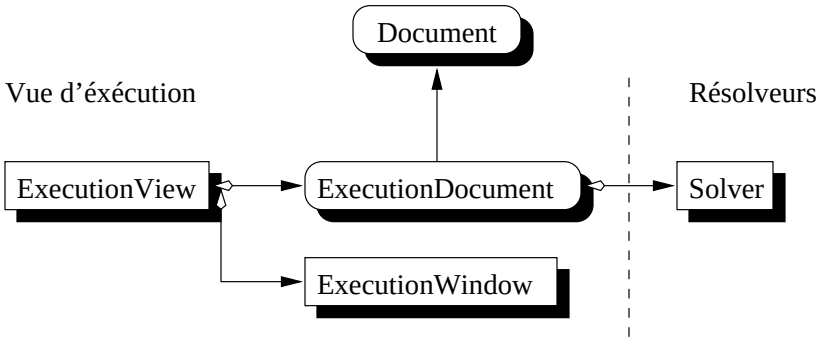

Figure 8 : Architecture globale de la vue d'exécution

3.2 Le document exécutable (*ExecutionDocument*)

Le document exécutable inclut les attributs et les actions permettant de jouer le document. Dans cette section, nous décrivons l'architecture mise en place (c.f. figure 9).

3.2.1 Objets média − composante logique

La classe *ExecutionBasicObject*, qui met en œuvre l'interface *ExecutionDocument*, regroupe les attributs globaux aux objets, comme la position. Elle gère aussi la progression temporelle de l'objet en utilisant la classe *Timer*. Cette architecture est conforme à l'architecture présentée dans [Sab98] qui spécifie un modèle objet pour les objets média de base de façon à assurer son extensibilité.

Ces attributs sont soit affectés directement par l'auteur, soit mis en relation avec d'autres attributs de même type ou de type différent (c.f. 2.1). En plus des attributs, chaque objet offre les méthodes d'accès et de modification des attributs (*set, get*), ainsi que les actions usuelles de progression (*play*, *stop*, *pause*, *resume*, etc.).

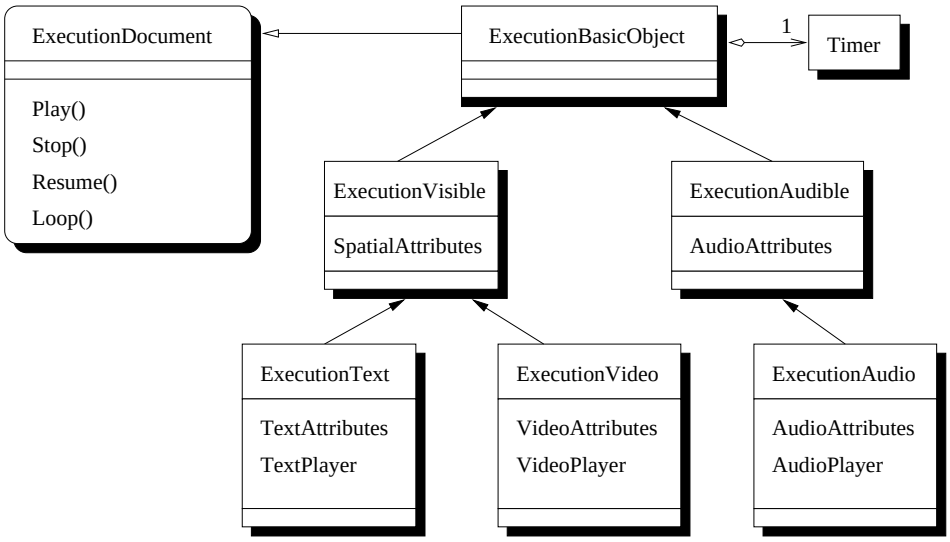

Figure 9 : Architecture des objets média

Pour maintenir la cohérence du jeu de relations spécifiées par l'auteur, il est nécessaire d'utiliser un résolveur de contraintes.

3.2.2 Résolveur de contraintes

Après une analyse de la littérature sur les résolveurs de contraintes ([Car97] [San93][San94]), nous pouvons les classer en deux catégories : ceux utilisant une approche locale, et ceux utilisant une approche globale. Cette analyse nous a permis de définir deux interfaces, *LocalSolvers* et *GlobalSolvers* (c.f figure 10), permettant de faire abstraction d'une quelconque implémentation d'un résolveur. L'objectif est de pouvoir tester plusieurs résolveurs. L'interface *Solver* réunit les actions communes aux deux types d'approches.

Comme la version précédente de Madeus utilisait un résolveur pour maintenir les relations portant sur l'attribut position, nous avons choisi de le réutiliser et de l'appliquer à l'ensemble des attributs. L'algorithme utilisé, appelé *Deltablue* [San93], fait partie de la classe de résolveur par propagation locale. Donc il met en œuvre l'interface *LocalSolvers*.

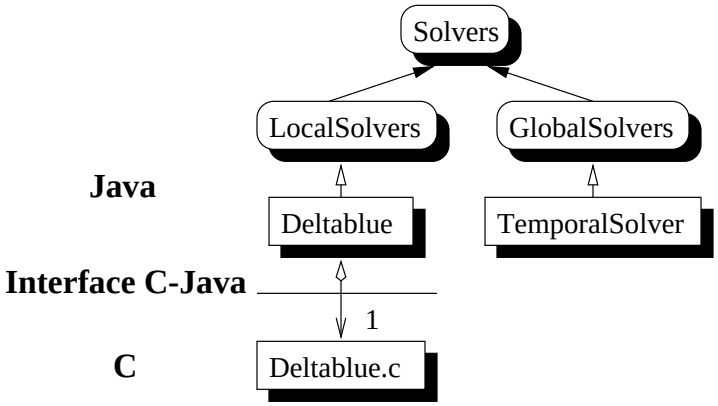

Figure 10 : Architecture des résolveurs

La mise en œuvre de *Deltablue*, écrit en langage C ANSI, a nécessité l'utilisation des mécanismes permettant d'intégrer dans un programme Java du code natif écrit en C. De plus, les limites de *Deltablue*, notamment son incapacité à gérer les cycles, ont été partiellement contournées. Ce travail, réalisé dans le cadre d'un stage ingénieur CNAM (c.f. [Car97]), a aussi été intégré dans Madeus 2.0. Comme tous les modules C ont été écrits en respectant la norme ANSI leur compilation ne pose aucun problème sur différentes plateformes utilisées (Linux, Solaris, Windows), et donc Madeus 2.0 ne souffre pas de problème de portabilité.

3.2.3 Expression spatio−temporelle

Nous avons vu que chaque objet est caractérisé par un ensemble de propriétés. L'introduction du spatio−temporel nécessite de mettre en place un mécanisme permettant de modifier dynamiquement la valeur des propriétés selon les spécifications de l'auteur. Dans Madeus 2.0, les propriétés des objets sont implantées par les classes ***Type****Attribut* (c.f figure 11) où *type* correspond aux types de base de Java (*integer*, *float*, etc.). Pour l'expression spatio−temporelle, ces classes sont étendues afin d'intégrer les méthodes de progression (*play*, *stop*, etc.) sur les propriétés. Cette progression est contrôlée par une fonction qui correspond à la spécification donnée par l'auteur (déplacement, zoom, etc.)..

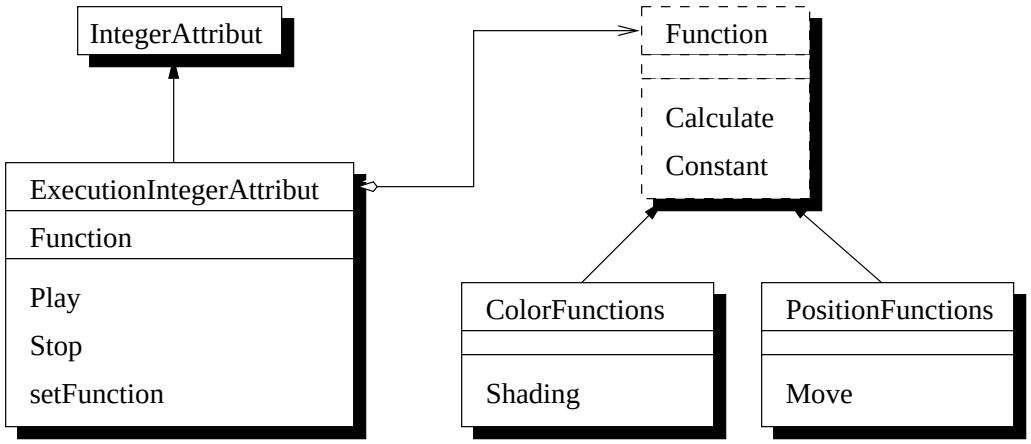

Figure 11 : Classes implémentant l'expression spatio−temporelle

3.3 La fenêtre d'exécution

La fenêtre d'exécution contient la partie visible et audible du document multimédia. Elle est constituée de plusieurs *ObjectPlayer* qui constituent la partie physique des objets média. La mise en œuvre repose sur le paquet JFC[(5)] pour les objets statiques (texte, image, etc.) et sur le paquet JMF[(6)] pour les objets dynamiques. On peut remarquer que cette architecture est le symétrique de la partie logique : chaque objet logique contient une référence vers un objet physique.

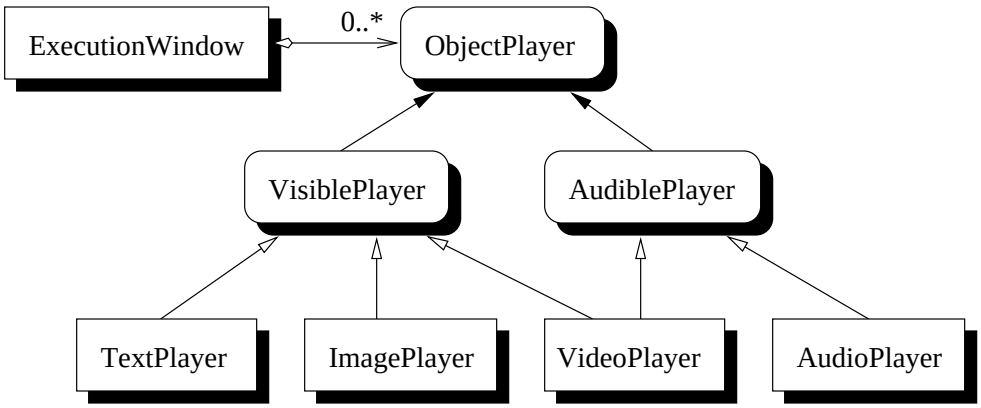

Figure 12 : Les objets média dans la fenêtre d'exécution

3.4 Bilan sur le système d'exécution

Le système d'exécution tel qu'il a été décrit dans les sections précédentes a été mis en œuvre dans Madeus 2.0 et est pleinement fonctionnel. Les sous−intervalles sont mieux pris en compte par rapport à la version 1.0, tant au niveau de la spécification que de l'exécution. Le code résultant correspond à envirion 8500 lignes.

À partir de cela, nous sommes apte pour aller plus loin en vérifiant la cohérence d'un document multimédia intégrant l'expression spatio−temporelle.

4 Vérification de la cohérence spatio−temporelle

Dans cette section, nous commençons par effectuer une analyse des conséquences de l'ajout d'informations spatio−temporelles dans un document multimédia en ce qui concerne la cohérence de ce document. De cette analyse, nous en déduisons une démarche pour ajouter de telles informations. Ensuite, nous verrons comment les résolveurs (spatial et temporel) utilisés dans Madeus peuvent répondre à la mise en œuvre de cette démarche et nous identifierons leurs limites.

4.1 Identification des incohérences

L'identification des incohérences s'effectue dans le cadre du problème du maintien de la cohérence, c'est−à−dire que la spécification est initialement cohérente et l'utilisateur augmente incrémentalement cette spécification. Le système doit alors vérifier si la spécification obtenue est toujours cohérente ou effectuer les opérations nécessaires pour qu'il reste cohérent.

Par définition, une spécification est incohérente lorsque le système ne peut pas trouver de solution pour exécuter le document. Nous dirons par la suite qu'une spécification est *faiblement* incohérente lorsque le système peut jouer un document malgré une partie inutile dans la spécification. Par exemple, si l'auteur spécifie une relation spatiale temporisée (c.f. 2.1) durant deux secondes entre deux objets et que la durée de ces objets est inférieure à dix secondes, alors la spécification est faiblement incohérente.

La suite de cette section est organisée comme suit : l'identification des incohérences potentielles est effectuée lors de la spécification d'un objet. Ensuite, nous étendons cette identification lorsque des relations entre deux objets sont spécifiées.

4.1.1 Mono−objet

Attributs spatio−temporels

L'introduction des attributs spatio−temporels implique qu'un attribut puisse être fixé de façon absolue. Par conséquent, il est nécessaire d'ajouter une contrainte sur la valeur de l'attribut, contrainte spécifiant que cette valeur est la seule possible pour un instant donné. Par exemple, la figure 13 illustre un cas d'incohérence et un cas de cohérence lorsqu'une relation d'alignement gauche est ajoutée

à la spécification. Le terme *ancré* pour un objet signifie que l'auteur a spécifié explicitement une position pour l'objet.

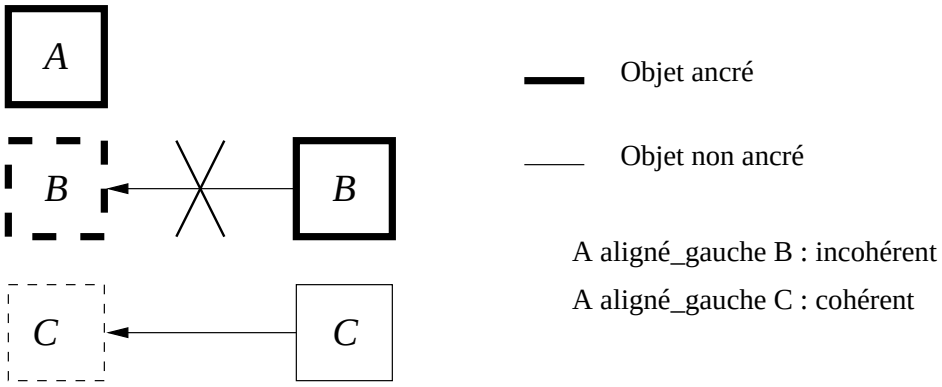

Figure 13 : Exemple d'incohérence sur les attributs spatio−temporels

Relations spatio−temporelles intra−objet

La seule vérification à effectuer porte sur l'exclusion mutuelle des sous−intervalles spécifiant une fonction pour un type d'attribut. En effet, il est incohérent de spécifier deux valeurs pour un attribut.

4.1.2 Multi−objets

Dans la suite, nous nous restreignons, sans perte de généralité, à des relations entre des objets. Nous ne considérons pas les relations entre sous−intervalles puisque ce type de relations est équivalent aux relations entre des intervalles (qui caractérisent les objets).

4.1.2.1 Relations spatiales en fonction du temps

Nous avons vu dans la section 2.1 que nous avons défini trois types de relations spatiales en fonction de leur durée de vie. Nous analysons ci−dessous les vérifications à effectuer lors de chacune d'elle.

Relations ponctuelles

C'est le cas le plus simple. À un instant donné, il faut vérifier que le système de contraintes avec l'ajout incrémental d'une relation spatiale ponctuelle n'introduit pas d'incohérences. Ce type de vérification existe dans la première version de Madeus. Elle est mise en œuvre grâce à l'algorithme DeltaBlue (c.f. [San93]) et des améliorations ont été apportées pour gérer certains cas de redondance (c.f. [Car97]). Par contre, il reste des cas où le système ne sait pas trouver de solution, alors qu'une solution existe (problème de cycle). Par exemple, si l'auteur spécifie une relation

d'alignement droit et une relation d'égale largeur entre deux objet *A* et *B*, alors on obtient un cycle de contraintes que DeltaBlue ne sait pas gérer.

Relations intemporelles

Contrairement au cas précédent, la contrainte porte sur toute la durée de vie des objets en relation. Or, les attributs spatiaux de ces objets peuvent potentiellement varier en fonction du temps (par exemple une fonction de déplacement sur l'attribut *Position*). Donc la contrainte est à vérifier non plus sur une valeur mais sur une fonction. Nous commençons par poser formellement le problème en donnant la solution générale. Ensuite nous effectuons une analyse de cas selon le nombre de contraintes spatiales et temporelles portées par les objets.

Problème formel et solution générale

De façon générale, soient *A* et *B* deux objets, et $f_a(t)$ (respectivement $f_b(t)$) la fonction définissant un attribut de l'objet *A* (resp. *B*), avec *t* une valeur comprise entre zéro et la durée de l'objet *A* (resp. *B*). L'instant initial de ces fonctions est l'instant de début de l'objet qu'elles caractérisent. Prenons le cas d'une relation d'alignement intemporelle entre ces deux objets qui, en terme de contraintes, se représente par une égalité de position entre ces deux objets. Deux cas se présentent :

- les deux objets commencent en même temps. Le système doit vérifier que $f_a(t) = f_b(t)$ pour tout $t \in [0, \Delta]$, avec Δ la durée totale de la relation (c.f. 2.1). Par conséquent, il faut vérifier que $f_a \approx f_b$ (f_a équivalent à f_b) ;
- les objets ne commencent pas en même temps. Dans ce cas, l'égalité à vérifier est $f_a(t + d) = f_b(t)$ avec d le délai entre les deux objets. Dans Madeus, ce délai est un intervalle de durée possible, mais pour simplifier notre discours nous considérons que le délai entre deux objets est une valeur.

En résumé, le système doit vérifier que pour tout $t \in [0, \Delta]$, $f_a(t + d) = f_b(t)$ avec d le délai entre *A* et *B*. La figure 14 illustre cette équation : si les deux courbes peuvent « se mapper » pendant un temps Δ, alors il y a égalité.

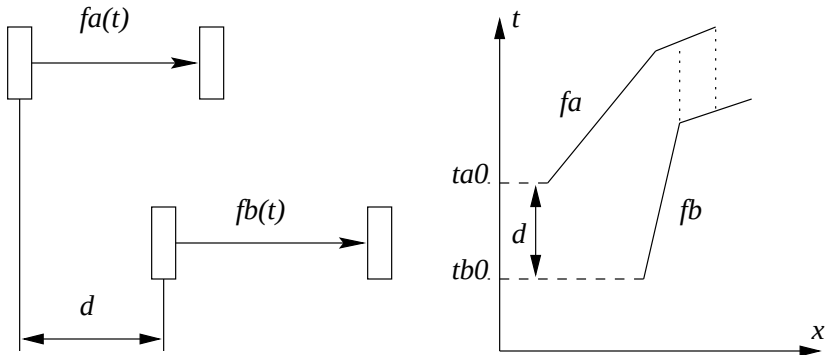

Figure 14 : Cas général

Analyse détaillée

Plus précisément, nous pouvons distinguer trois cas selon le nombre de contraintes spatiales existantes pour un attribut, c'est−à−dire soit les deux objets ont une contrainte spatiale, soit un des deux a une contrainte spatiale, soit aucun des deux objets n'en a une. Un objet a une contrainte spatiale lorsqu'il est caractérisé par une fonction spatiale. Par exemple, si l'objet A est contraint spatialement, alors f_a existe. L'origine de cette contrainte est dûe soit à une spécification absolue (f_a = 10) ou via une fonction (f_a= 10.t + 30), ou soit via une relation (A *aligné_gauche* B). Dans tous les cas, cette spécification est effectuée par l'auteur, c'est−à−dire que les valeurs par défaut ne sont pas considérées comme originaires d'une contrainte spatiale.

Pour chacun de ces trois cas, il y a plusieurs autres cas selon le nombre de contraintes temporelles portées par ces objets. De la même façon, un objet est contraint temporellement lorsqu'il existe une spécification de l'auteur plaçant explicitement cet objet dans le temps.

Voici les trois cas selon le nombre de contraintes spatiales :

A et B contraint spatialement

les deux objets mis en relation sont contraints spatialement, c'est−à−dire que f_a et f_b existent.

- les objets sont contraints temporellement, donc d = d_r n'est plus un paramètre. Il faut vérifier que $f_a(t + d_r) = f_b(t)$ pour $t \in [0, \Delta]$ et d_r le délai entre les objets A et B tel que défini par la relations temporelles entre A et B. L'ajout d'une relation spatiale intemporelle entre des objets déjà très contraints est peu réaliste. Par conséquent, ce genre de vérification est relativement rare ;

- un des objets n'est pas contraint temporellement : il faut vérifier qu'il existe un d tel que $f_a(t + d) = f_b(t)$. Si ce d existe, alors l'objet non contraint temporellement le devient. Eventuellement, un ensemble de valeurs pour d permet cette propriété.

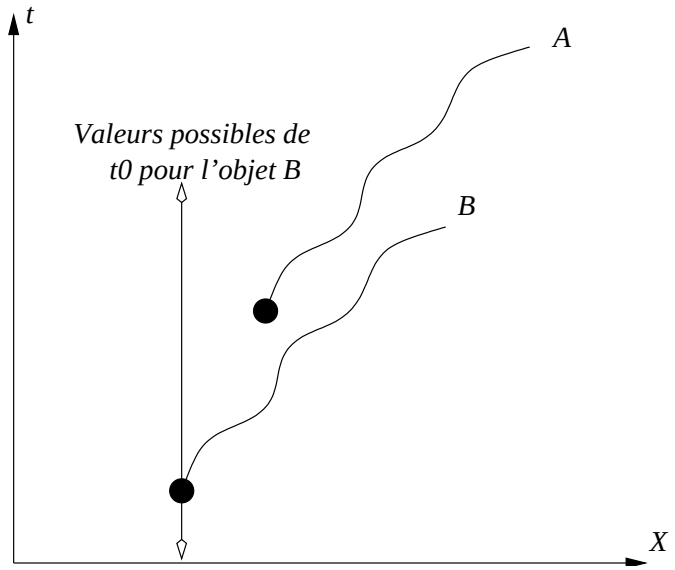

Figure 15 : Cas de cohérence lorsque A et B sont contraints spatialement et ont un degré de liberté sur l'axe temporel

A contraint spatialement

Un seul des attributs est contraint. Supposons que ce soit un attribut de l'objet A, c'est−à−dire que la fonction f_a existe mais pas la fonction f_b. Dans ce cas, nous sommes sûrs de trouver une solution valide en fixant f_b par une fonction adéquate.

- si les deux objets sont contraints temporellement, alors la fonction f_b de l'objet B doit être égale à $f_a(t + d)$ avec d le délai entre les deux objets ;
- si l'objet A n'est pas contraint temporellement, mais l'objet B l'est, alors $f_b = f_a(t + d)$ avec d le délai entre A et B. Ce délai est un intervalle de temps qui dépend du début de B, de la durée de A et B et du début et de la fin du composite. Cet intervalle est le suivant : I_d = [X..Y] avec X = début(B) et X = ((fin(Composite) − durée(A) − début(B)) ;
- si l'objet A est contraint temporellement, mais pas l'objet B, il y a encore plus de possibilités valides. Il suffit de prendre un f_b tel que $f_b = f_a(t + d)$ avec d compris entre zéro et la valeur maximum permise par le composite ;

- si aucun des deux objets n'est contraint temporellement, nous retombons sur le cas précédent sauf que ce cas offre plus de liberté puisque le début de B n'est pas fixé. Les valeurs permises sur le délai sont délimités par l'intervalle I_d= [X..Y] avec X = fin(Composite) − Durée(B) et Y = (fin(Composite) − durée(A)].

Ni A et B ne sont constraints spatialement

Les deux objets mis en relation ne sont pas contraints spatialement. La contrainte $f_a(t + d) = f_b(t)$ est alors ajoutée au système de contraintes, en sachant que les fonctions f_a et f_b sont des fonctions par défaut et donc elles peuvent être modifiées.

Relations temporisées

La troisième vérification à effectuer porte sur les relations spatiales temporisées. Cette vérification sur ce type de relation est un cas particulier du précédent en restreignant le domaine de t à la durée de la relation.

En résumé, il faut que le résolveur soit capable d'effectuer des comparaisons entre des fonctions (égalité, inférieur, supérieur, etc.), en sachant que ces comparaisons dépendent du domaine des fonctions (domaine de t) et de la translation temporelle (l'argument d) entre ces fonctions. Cette translation n'est pas fixe, mais correspond à un ensemble de valeurs.

4.1.2.2 Relations temporelles en fonction du spatial

Ce type de relation permet de placer des objets temporellement en fonction de la valeur des attributs spatiaux. Durant mon stage de DEA, j'ai montré que ces relations se réduisaient à de simples relations temporelles. Il suffit de calculer l'attribut temporel impliqué dans la relation.

Par exemple, la relation *A starts((X = 50)) B*, qui signifie que l'objet A commence lorsque l'objet B est à la position verticale 50, ce traduit en terme d'équation comme suit :

$$\text{Début}(A) = t \,/\, (X_b = 50) \Leftrightarrow \text{Début}(A) = t \,/\, (\, f_a(t) = 50\,)$$

$$\Leftrightarrow \text{Début}(A) = t \,/\, t = f^{-1}(50)$$

Il faut donc, dans un premier temps, calculer la fonction inverse, puis intégrer la valeur trouvée dans le vérificateur temporel. Si aucune valeur n'existe, alors le système est incohérent. Par contre, si plusieurs valeurs existent, il est nécessaire que l'utilisateur

précise laquelle utiliser. Par exemple, pour un cercle il doit préciser le numéro du tour pendant lequel la valeur apparaîtra.

4.2 Intégration dans l'existant

La vérification spatio−temporelle utilise à la fois les services du vérificateur temporel et du vérificateur spatial. Dans Madeus 2.0, le vérificateur temporel est PC−2 (c.f. [Mac77]). Il fait partie de la classe des STP (Simple Temporal Problem). Les algorithmes appartenant à cette classe ne permettent pas d'exprimer l'exclusion mutuelle entre sous−intervalles. Ils ne permettent pas non plus d'avoir une union de plusieurs intervalles de durée pour un intervalle. Pour le premier problème, une solution est d'obliger l'auteur à spécifier explicitement des relations temporelles entre les sous−intervalles.

Le vérificateur spatial, mis en œuvre par l'algorithme Deltablue, est un vérificateur très général. L'implantation actuelle permet de résoudre des contraintes portant sur des entiers. Mais il est possible de l'étendre à des fonctions. La difficulté réside dans la comparaison de fonctions (c.f. 4.1.1) qui doit calculer toutes les solutions possibles pour que la comparaison soit valide. De plus, le résolveur doit être capable d'ajouter et d'enlever dynamiquement des contraintes.

Comme Madeus est un système interactif, il est nécessaire de mettre en place des algorithmes performants, tant au niveau du temps d'exécution mais aussi au niveau des capacités à résoudre un système de contraintes. À notre connaissance, il n'existe pas d'algorithme permettant de résoudre des contraintes portant sur des fonctions dépendantes du temps. Les travaux F. Pachet et O. Delerue (c.f. [Pac98]) nécessitant un résolveur de contraintes non linéaires, non fonctionnelles (par exemple les inégalités) et gérant d'une façon particulière les cycles, l'ont conduit à développer un résolveur par propagation locale ad−hoc.

Cassowary (c.f [Bad98]), développé par l'université de Washington à Seattle, est le successeur de Deltablue. Il permet de résoudre des systèmes de contraintes linéaires, d'inégalités et gère les cycles. Cet algorithme pourra être expérimenté dans Madeus 2.0 pour assurer une meilleur gestion des contraintes spatiales (notamment pour les cycles). Cependant, cela ne sera peut être pas suffisant notamment pour des raisons d'efficacité lors des comparaisons de fonctions. Ainsi, nous pensons que la vérification de l'expression spatio−temporelle dans Madeus nécessitera de concevoir et de mettre en œuvre d'un algorithme ad−hoc (éventuellement à partir de Cassowary), reposant sur la démarche définie dans la section 4.1.2.1.

5 Conclusion

5.1 Synthèse

Dans ce rapport nous avons décrit les travaux effectués dans le cadre d'un magistère troisième année. Les objectifs principaux ont été d'intégrer l'expression spatio−temporelle dans la version 2.0 de Madeus, et pour cela, nous avons commencer par définir une syntaxe pour écrire un document multimédia. Cette syntaxe intègre l'expression de Madeus 1.0 à laquelle l'expression spatio−temporelle est ajoutée de façon homogène. Le choix de cette syntaxe s'est orienté vers le standard XML pour son pouvoir d'extensibilité, sa lisibilité et le nombre d'outils existant, comme notamment des parsers.

Ensuite, nous avons défini une architecture logicielle pour exécuter un document multimédia intégrant l'expression spatio−temporelle. Cette architecture s'insère dans l'architecture globale de Madeus 2.0 puisqu'elle repose sur la notion de vue particulière d'un document multimédia. Nous avons donc défini une vue d'exécution composée d'un document exécutable, contenant les informations nécessaires à l'exécution comme les attributs spatiaux, l'horloge ou encore un résolveur spatial. Le deuxième composant de la vue est la fenêtre d'exécution qui permet de jouer physiquement les objets média. Ces deux composants intègrent l'expression spatio−temporelle.

Enfin, nous avons effectué une analyse des différents cas d'incohérence lors de la spécification incrémentale par l'expression spatio−temporelle. Grâce à cette analyse, nous avons proposé une démarche prenant en compte les différents cas possibles pour soit ajouter des contraintes implicites, soit effectuer des vérifications. Puis, nous avons vu dans quelle mesure il était possible d'effectuer la vérification spatio−temporelle avec les vérificateurs PC−2 (temporel) et DeltaBlue (spatial) de Madeus. À partir d'un rapide état de l'art, nous proposons enfin une piste pour étendre ces techniques.

5.2 Perspectives envisagées

À court terme, la méthode proposée pour vérifier l'expression spatio−temporelle est à spécifier avec précision afin de pouvoir la mettre en œuvre et la valider. Ce type de vérification est nouveau, assez difficile, et lors des extensions de l'expression d'un document multimédia d'autres besoins de vérifications vont apparaître. Cela pose le problème de savoir jusqu'où peut−on aller dans la spécification déclarative d'applications fortement interactives et dynamiques.

La mise en œuvre de la vue d'exécution a posé des problèmes au niveau de la qualité de service. Plusieurs axes de recherche sont envisagés à ce niveau [Haf98]:

- définition d'une architecture logicielle basée sur une horloge logique pour permettre une meilleure gestion des processus léger ;
- meilleure gestion des ressources physiques, en tirant profit par exemple des possibilités d'accélération matérielle ;
- prédiction pour le chargement des ressources ;
- formatage dynamique, c'est−à−dire adapter la durée d'un objet en fonction de l'accès aux ressources et aussi de palier au problème d'indéterminisme.

5.3 Remerciements

Je remercie tout d'abord Claude Puech pour m'avoir proposé de faire un magistère et de bien vouloir faire partie de ce jury.

Je remercie Vincent Quint pour m'avoir accueilli au sein du projet Opéra.

Je remercie particulièrement Cécile Roisin pour m'avoir accueilli et pour toute l'aide qu'elle m'a apportée et aussi sa patience lors des nombreuses relectures de ce rapport.

Je remercie Muriel Jourdan pour ses conseils lors des relectures.

Je remercie toute l'équipe Madeus, Laurent, Frédéric, Loay, Yves et surtout Nabil Layaïda sans qui ce rapport n'aurait jamais existé.

Enfin, un remerciement spécial, pour sa patience et ses encouragements, à ma chère Nadia.

Annexe A DTD Madeus

```
<!--============================================-->
<!--======================== Generique entities -->
<!--============================================-->
<!ENTITY % link_properties
                 "a CDATA #IMPLIED
                  anchor CDATA #IMPLIED"          >

<!ENTITY % position_properties
                 "Left CDATA #IMPLIED
                  Top  CDATA #IMPLIED"            >
<!ENTITY % position_functions "(segment)">

<!ENTITY % size_properties
                 "Width  CDATA #IMPLIED
                  Height CDATA #IMPLIED"          >

<!ENTITY % color_properties
                 "Red   CDATA #IMPLIED
                  Green CDATA #IMPLIED
                  Blue  CDATA #IMPLIED"           >

<!ENTITY % font_properties
                 "FontFamily CDATA #IMPLIED
                  FontSize   CDATA #IMPLIED
                  FontStyle  CDATA #IMPLIED"      >

<!ENTITY % global_properties
                 "Name     CDATA #REQUIRED
                  Source   CDATA #IMPLIED
                  Value    CDATA #IMPLIED
                  Duration CDATA #IMPLIED"        >

<!ENTITY % all_properties
                "%global_properties
                  %position_properties;
                  %link_properties;
                  %font_properties;)"             >

<!ENTITY % objects "(text | picture | external)"   >

<!--============================================-->
<!--================== Generally useful entities -->
<!--============================================-->


<!--============================================-->
<!--================================== Document -->
<!--============================================-->
```

```
<!ELEMENT Madeus (%objects; | composite)>
<!ATTLIST Madeus
          Name CDATA #REQUIRED
          Version CDATA #REQUIRED>

<!--================================================-->
<!--================================= Text object -->
<!--================================================-->
<!ENTITY % text_properties "%position_properties;
                            %font_properties;
                            %color_properties;
                            %link_properties;"     >
<!ENTITY % text_functions  "%position_functions;
                            %font_functions;
                            %color_functions;"     >
<!ELEMENT text (text_intervals?, relations?)       >
<!ATTLIST text %global_properties;
               %text_properties;                   >
<!ELEMENT text_intervals (text_interval+)          >
<!ELEMENT text_interval (%text_functions;)*        >
<!ATTLIST text_interval
          %global_properties;
           %text_properties;                       >

<!--================================================-->
<!--============================== Picture object -->
<!--================================================-->
<!ENTITY % picture_properties "%position_properties;
                               %size_properties;"  >
<!ENTITY % picture_functions "%position_functions;">
<!ELEMENT picture (picture_intervals?, relations?) >
<!ATTLIST picture %global_properties;
                  %picture_properties;             >
<!ELEMENT picture_intervals (picture_interval+)    >
<!ELEMENT picture_interval (%picture_functions;)*  >
<!ATTLIST picture_interval
          %global_properties;
           %picture_properties;                    >

<!--================================================-->
<!--============================ Composite object -->
<!--================================================-->
<!ELEMENT composite (default , (%objects; |
                     composite)+, relations)       >
<!ATTLIST composite
           name CDATA #REQUIRED
           %position_properties;                   >
<!ELEMENT default EMPTY                            >
<!ATTLIST default
           %all_properties;                        >

<!--================================================-->
<!--================================== Functions -->
<!--================================================-->
```

```
<!ELEMENT segment EMPTY                              >
<!ATTLIST segment
 src_left CDATA #IMPLIED
 src_top CDATA #IMPLIED
 dst_left CDATA #IMPLIED
 dst_top CDATA #IMPLIED                              >

<!--============================================-->
<!--================================ Relations -->
<!--============================================-->
<!ELEMENT relations ((temporal, spatial?) |
                     spatial)                        >

<!--============================================-->
<!--================================ temporal -->
<!--============================================-->
<!ENTITY % operands "
 interval1 CDATA #REQUIRED
 interval2 CDATA #REQUIRED"                          >
<!ENTITY % operands_delay "
 interval1 CDATA #REQUIRED
 interval2 CDATA #REQUIRED
 delay CDATA #IMPLIED"                               >
<!ELEMENT temporal (equals | meets | met_by |
                    finishes | finished_by |
                    starts | started_by | kills |
                    killed_by | parmin | lipsync |
                    lipsync_by | before |
                    after | during | contains |
                    overlaps | overlaps_by |
                    begins | ends)*                  >


<!ELEMENT equals EMPTY                               >
<!ATTLIST equals %operands;                          >

<!ELEMENT meets EMPTY                                >
<!ATTLIST meets %operands;                           >

<!ELEMENT met_by EMPTY                               >
<!ATTLIST met_by %operands;                          >

<!ELEMENT finishes EMPTY                             >
<!ATTLIST finishes %operands;                        >

<!ELEMENT finished_by EMPTY                          >
<!ATTLIST finished_by %operands;                     >

<!ELEMENT starts EMPTY                               >
<!ATTLIST starts %operands;                          >

<!ELEMENT started_by EMPTY                           >
<!ATTLIST started_by %operands;                      >
```

```
<!ELEMENT kills EMPTY                                >
<!ATTLIST kills %operands;                           >

<!ELEMENT killed_by EMPTY                            >
<!ATTLIST killed_by %operands;                       >

<!ELEMENT parmin EMPTY                               >
<!ATTLIST parmin %operands;                          >

<!ELEMENT lipsync EMPTY                              >
<!ATTLIST lipsync %operands;                         >

<!ELEMENT lipsync_by EMPTY                           >
<!ATTLIST lipsync_by %operands;                      >

<!ELEMENT before EMPTY                               >
<!ATTLIST before %operands_delay;                    >

<!ELEMENT after EMPTY                                >
<!ATTLIST after %operands_delay;                     >

<!ELEMENT during EMPTY                               >
<!ATTLIST during %operands_delay;                    >

<!ELEMENT contains EMPTY                             >
<!ATTLIST contains %operands_delay;                  >

<!ELEMENT overlaps EMPTY                             >
<!ATTLIST overlaps %operands_delay;                  >

<!ELEMENT overlaps_by EMPTY                          >
<!ATTLIST overlaps_by %operands_delay;               >

<!ELEMENT begins EMPTY                               >
<!ATTLIST begins %operands_delay;                    >

<!ELEMENT ends EMPTY                                 >
<!ATTLIST ends %operands_delay;                      >

<!--============================================-->
<!--================================== spatial -->
<!--============================================-->

<!ENTITY % operands_dist "
 interval1 CDATA #REQUIRED
 interval2 CDATA #REQUIRED
 distance CDATA #IMPLIED"                            >
<!ELEMENT spatial (left_align | center_v |
                   right_align | left_spacing |
                   left_ident | right_ident |
                   right_spacing | top_align |
                   center_h | bottom_align |
                   top_spacing | top_ident |
```

```
                    bottom_ident)*                >

<!ELEMENT left_align EMPTY                        >
<!ATTLIST left_align %operands;                   >

<!ELEMENT center_v EMPTY                          >
<!ATTLIST center_v %operands;                     >

<!ELEMENT right_align EMPTY                       >
<!ATTLIST right_align %operands;                  >

<!ELEMENT left_spacing EMPTY                      >
<!ATTLIST left_spacing %operands_dist;            >

<!ELEMENT left_ident EMPTY                        >
<!ATTLIST left_ident %operands_dist;              >

<!ELEMENT right_ident EMPTY                       >
<!ATTLIST right_ident %operands_dist;             >

<!ELEMENT right_spacing EMPTY                     >
<!ATTLIST right_spacing %operands_dist;           >

<!ELEMENT top_align EMPTY                         >
<!ATTLIST top_align %operands;                    >

<!ELEMENT center_h EMPTY                          >
<!ATTLIST center_h %operands;                     >

<!ELEMENT bottom_align EMPTY                      >
<!ATTLIST bottom_align %operands;                 >

<!ELEMENT top_spacing EMPTY                       >
<!ATTLIST top_spacing %operands_dist;             >

<!ELEMENT top_ident EMPTY                         >
<!ATTLIST top_ident %operands_dist;               >

<!ELEMENT bottom_ident EMPTY                      >
<!ATTLIST bottom_ident %operands_dist;            >

<!ELEMENT bottom_spacing EMPTY                    >
<!ATTLIST bottom_spacing %operands_dist;          >
```

Annexe B Notation UML

UML (Unified Modeling Language [Boo98]) est un langage pour spécificier, visualiser, construire et documenter les objets d'un logiciel. Cette annexe n'a pas l'ambition de présenter ce standard mais simplement quelques diagrammes nécessaires à la compréhension des schémas de ce rapport.

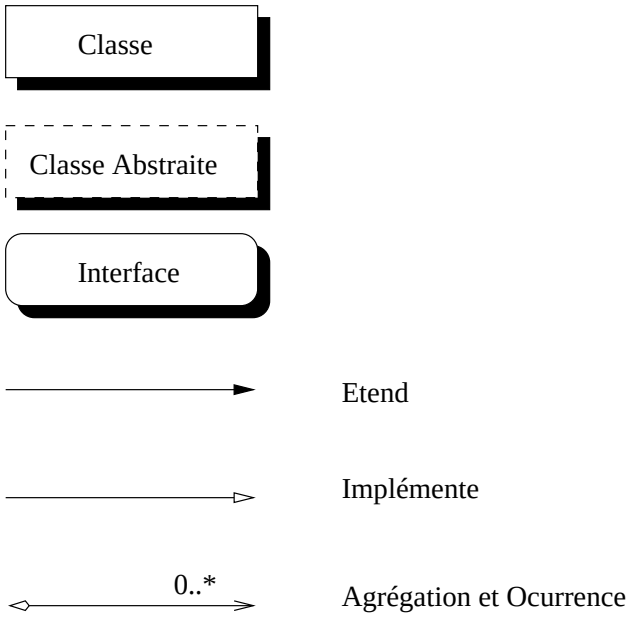

Figure 16 : Notation UML

Bibliographie

[Bad98] G.J. Badros, A. Borning, ‘‘The Cassowary Linear Arithmetic Constraint Solving Algorithm : Interface and Implementation’’, *Technical Report UW−CSE University of Washington*, Juin 1998.

[Boo98] G. Booch, I. Jacobson, J. Rumbauch, ‘‘Unified Modelling Language’’, *http://www.rational.com/uml/index.shtml*, 1998.

[Car97] L. Carcone, *Formatage spatial dans un environnement d’édition/présentation de documents multimédia*, Mémoire, Conservatoire National des Arts et Métiers, Décembre 1997.

[Cou87] J.Coutaz, ‘‘PAC : an Implementation Model for Dialog Design’’, *Proceedings of Interact’87*, pp. 431−436, Septembre 1987.

[Dec96] D.Decouchant, M. Roméro, *Structured and Distributed Cooperative Editing in a Large Scale Network, Groupware and Authoring*, Chapter 13, pp. 265−295, Academic Press Londres, Angleterre, May 1996.

[Haf98] A. Hafid, G. Bochmann, R. Dssouli, ‘‘Distributed multimédia application and quality of service : a review’’, *Electronic Journal on Networks and Distributed Processing*, (n°6), pp. 1−50, 1998.

[ISO86] I.S.O, *Information processing − Text and office systems − Standard Generalized Markup Language (SGML)*, num. ISO 8879, 1986.

[Jou98a] M. Jourdan, N. Layaida, C. Roisin, L. Sabry−IsmailL. Tardif, ‘‘Madeus, an Authoring Environnement for Interactive Multimedia Document’’, *ACM Multimedia’98*, pp. 267−272, Septembre 1998.

[Jou98b] M. Jourdan, C. Roisin, L. Tardif, ‘‘Multiviews Interfaces for Multimedia Authoring Environnement’’, *Proceeding of the 54th Conference on Multimedia Modeling*, Novembre 1998.

[Lay97] N. Layaïda, *Madeus : Système d’édition et de présentation de documents structurés multimédia*, Thèse, Université Joseph Fourier, Juin 1997.

[Mac77] A.K Mackworth, ‘‘Consistency in Networks of Relations’’, *Artificial Intelligence*, Volume 8(N°1), pp. 99−118, 1977.

[Pac98] F. Pachet, O. Delerue, ‘‘MidiSpace : a Temporal Constraint−Based Music Spatializer’’, *ACM Multimedia’98*, pp. 351−359, 1998.

[Qui87] V. Quint, *"Une approche de l'édition structurée des documents"*, Thèse d'état, Université scientifique technologique et médicale de Grenoble, Mai 1987.

[Sab98] L.Sabry−Ismail, R.Guetari, ‘‘Le modèle objet Madeus’’, *L'objet : Les représentations par Objets en Conception*, Volume 4(N°2), 1998.

[San93] M. Sannella, J. Maloney, B. Freeman−Benson, and A. Borning, ‘‘Multi−way versus One−way in User Interfaces : Experience with the DeltaBlue Algorithm’’, *Software−Pratice and Experience*, Vol.32(5), pp. 529−566, Mai 1993.

[San94] M. Sannella, ‘‘The SkyBlue Constraint solver and Its Applications’’, *In Saraswat ans Van Hentenryck editors, Proceeding of the 1993 Workshop on Principles and Practice of Constraint Programming*, MIT Press, 1994.

[Tig97] J.Tigue, T.Bray, ‘‘SAX 1.0: The Simple API for XML’’, *http://www.microstar.com/XML/SAX/*, December 1997.

[Vil98] L.Villard, *Spécification de relations spatio−temporelles dans un document multimédia*, DEA Informatique, Université Joseph Fourier, Grenoble, Juin 1998.

[W3C98a] W3C, ‘‘Cascading Style Sheet, level 2’’, *http://www.w3c.org/TR/REC−CSS2*, Janvier 1998.

[W3C98b] W3C, ‘‘Extensible Markup Language (XML)’’, *http://www.w3.org/TR/REC−xml* , Mai 1998.

[W3C98c] W3C, ‘‘Document Object Model Specification’’, *http://www.w3c.org/TR/WD−DOM*, Juillet 1998.