MASTER EN SCIENCES INFORMATIQUES

Systèmes de contraintes temporelles appliqués à l'édition de documents multimédia structurés

Chisogne, Christophe

Award date:
1999

Facultés Universitaires Notre-Dame de la Paix
Namur

Année académique
1998-1999

Systèmes de contraintes temporelles appliqués à l'édition de documents multimédia structurés

Christophe Chisogne

Mémoire présenté en vue de l'obtention du grade de
Maître en Informatique

Institut d'Informatique

Rue GrandGagnage, 21 • B-5000 Namur (Belgium)

Avant-propos

Résumé

Ce document s'intéresse à l'édition temporelle de documents multimédia structurés.

Nous procédons d'abord à un état de l'art dans le domaine de l'édition temporelle de documents multimédia. Nous en dégageons deux classes d'approches. La première est une classe d'approches opérationnelles. L'auteur définit directement l'exécution de la présentation de son document (placement et durée des éléments). La seconde est une classe d'approches déclaratives. L'auteur décrit des objets et les relations temporelles les liant. Un système de contraintes temporelles permet une bonne flexibilité lors de la description du scénario temporel d'un document. Le logiciel d'édition est ensuite chargé du calcul de la durée et des placements temporels de chaque élément.

Les approches déclaratives s'avèrent plus intéressantes du point de vue de l'auteur d'un document multimédia. Un logiciel développé à l'INRIA de Grenoble, Madison, utilise cette approche au moyen d'un langage ad hoc de description de documents, Madeus.

Les réseaux de contraintes posent la double question de l'existence et du calcul d'une solution. L'éditeur doit maintenir l'existence d'une solution à chaque opération d'édition. Cette opération de vérification de cohérence est assurée par un algorithme classique adapté aux circonstances (PC2). L'éditeur doit calculer au moins une solution satisfaisant toutes les contraintes temporelles. C'est l'opération de formatage, qui consiste à calculer automatiquement les placements (spatio-)temporels des éléments d'un document lors de sa présentation à l'écran. Cette phase réutilise PC2.

Nous aborderons plus en détail l'algorithme PC2 et les difficultés de son implémentation. Nous terminerons en soulignant un aspect non géré par PC2 : les intervalles de durée à caractère incontrôlable. Ces derniers sont indispensables pour fournir aux auteurs un outil logiciel suffisamment intéressant pour retenir leur attention.

Remerciements

Tout d'abord, je tenais à remercier tout particulièrement Laurent Tardif et mon père d'avoir accepté de sacrifier une partie de leur temps précieux sur l'autel des relectures de mémoire. Laurent a revu le fonds tandis que mon père a vérifié l'orthographe.[1]

Je remercie également l'équipe de l'INRIA à Grenoble, entourée de beaux paysages montagneux, où j'ai effectué mon stage au sein d'une équipe jeune et dynamique, Merci donc à Stéphane, Laurent, Lionel , Laurent, Nabyl, Loay, Frédéric, Muriel, Cécile, Vincent et ceux que j'oublie ;-). Ce fut l'occasion pour moi d'apprendre pas mal de choses sur le terrain, comme

1. Cela ne devait pas être facile pour un français de corriger des "fautes de belge" ;-)

le développement logiciel simultané en groupe de travail, avec contrôle de version CVS. J'y ai découvert aussi une dimension "réseau" importante, où je disposais d'un compte sur une machine, utilisais les fichiers d'une autre, et travaillais devant une troisième où tournait un serveur X, le tout de manière transparente. Il est vrai que travailler dans un environnement Linux bien administré est terriblement agréable.

Je me dois de remercier certains membres de l'institut d'informatique des FUNDP à Namur. Je pense en particulier aux deux chercheurs du projet EMIM, Benjamin et Laurent. Je remercie également mes promoteurs, messieurs Fichefet et Leclerc.

Ensuite, je tenais à remercier tout ceux qui ont permis à ce mémoire de figurer un jour sur support papier. Je pense à Linus Torvalds, auteur de mon système d'exploitation favori, Linux. Je pense aussi à Richard Stallman, à qui nous devons, outre le projet GNU et la FSF (Free Software Foundation), un des meilleurs compilateurs de langage C au monde (gcc) et le plus versatile des éditeur de texte, Emacs, tour à tour éditeur de texte, lecteur de mail, psychiâtre rogérien, browser web, comparateur de répertoires, ou même Tétris. Je n'oublie pas Donald Knuth, ses ouvrages de qualité à propos d'algorithmique et son implémentation de TEX, ce splendide formateur de documents de très haute qualité. Sans lui, l'impression de documents n'aurait pas si fière allure. Merci également à Leslie Lamport, auteur du langage de macros LATEX, d'avoir mis cette puissance à la portée des simples mortels. Je remercie également toute l'équipe de développeurs C++ ayant donné naissance début 1999 à LYX, cette interface graphique bien pratique, qui donne un aspect plus convivial à ce langage a priori rébarbatif qu'est LATEX 2_ε. Je me devais de remercier ces illustres personnages, qui ont réalisé l'exploit de mettre un travail titanesque à disposition d'autrui, et cela de manière purement gratuite!

Enfin, last but not least, je remercie ma famille, qui a eu le courage de me supporter pendant la laborieuse période de rédaction du présent mémoire.

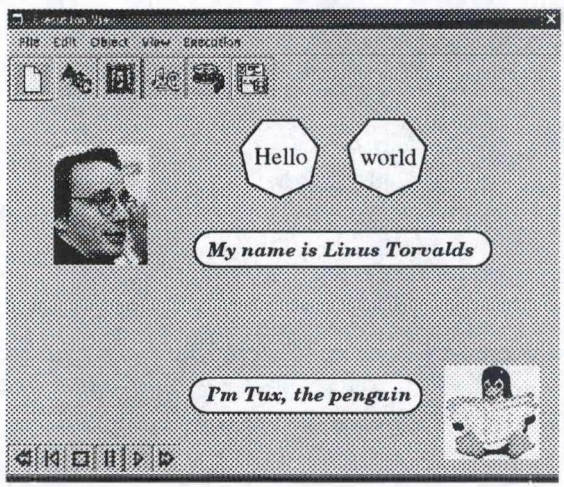


Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Introduction

Un thème à la mode : le multimédia

Ce mémoire traite des documents de type *multimédia*. L'informatique est une science en perpétuelle évolution. La banalisation de l'informatique, la diminution drastique des prix et une augmentation considérable de la puissance des ordinateurs personnels ont ouvert la voie à de nouvelles applications. Parmi elles, nous pouvons citer le thème du multimédia, toujours à la mode aujourd'hui. Le projet Cyberécoles appelle à la création de centres CyberMedias. Quant à la RTBF, elle annonçait en ce 8 août 1999 un projet américain de 6 milliards de francs pour rénover l'atomium de Bruxelles : le but serait d'installer un *hyperport* [1], un centre technologique dédié au multimédia et aux nouvelles technologies de l'information et de la communication.

Les documents structurés

Ce mémoire traite des documents multimédia ayant un caractère particulier : ils sont *structurés*. Ils peuvent se représenter sous forme d'arbre hiérarchique. Ceci permet de faire référence à leurs éléments constitutifs. Une autre caractéristique souvent associée est la notion de lien hypertexte. Ces liens référencent des éléments (images, autres documents), comme s'ils faisaient partie du document initial. Ainsi, nous pouvons réutiliser certaines parties de documents dans plusieurs autres. Ces documents sont décrits par des langages standardisés de description de documents, comme LaTeX, HTML, SMIL ou autres. Il existe même des métalangages qui permettent de définir des langages de description de document. C'est le cas de SGML et de XML. Ce dernier est une partie simple de SGML destinée à une implémentation plus aisée. C'est une des missions du W3C (`http://www.w3c.org`) de définir et standardiser tous ces langages et métalangages. J'aurai l'occasion d'y revenir plus loin dans ce mémoire. La capacité de réutilisation et la simplicité de HTML explique sans doute son succès auprès des internautes. Une dernière caractéristique de ces documents structurés est la possibilité de traiter de grandes masses de documents (dans des langues éventuellement différentes) à l'aide de traitements largement automatisés. Le projet Opera de l'INRIA [2] est entièrement dédié au

1. `http://www.rtbf.be/info/jt/1999_08_05/280.html`

2. Il s'agit de l'Institut National de Recherche en Informatique et en Automatique, organisme français. C'est à l'INRIA que nous devons, entre autres, le fameux langage Caml, et son successeur, Objective Caml.

thème des documents structurés, depuis longtemps déjà. Pour preuve, l'éditeur Thot qui y a été développé à l'initiative de Vincent Quint.

La quatrième dimension

Ce mémoire s'intéresse aux documents multimédia structurés de nature temporelle. Il existait hier encore trois axes indépendants pour la description classique des documents électroniques et structurés :

- la dimension logique : la structure hiérarchique du document, évoquée ci-dessus. Ainsi, XML décrit des classes de documents et leur décomposition en éléments constitutifs.
- la dimension sémantique : les hyperliens (annotations, renvois), à la manière du web. Par exemple, HTML décrit des renvois internes (vers une partie du document) et des renvois externes (vers d'autres documents). D'autres standards permettent même de lier plusieurs cibles à un même lien. Les hyperliens s'étendent également dans le temps : ils peuvent par exemple faire référence à un instant précis du document. Cette dimension garantit une certaine interactivité.
- la dimension spatiale : c'est la répartition spatiale des éléments sur le support (papier/écran). Un même document (contenu) peut être présenté de diverses manières, selon des feuilles de style. John Bozak, inventeur de XML, a illustré de manière étonnante cette capacité, lors d'une conférence à l'INRIA fin 1998.

Aujourd'hui, un quatrième axe vient s'ajouter :

- la dimension temporelle : il s'agit du scénario temporel associé au document. Du statique, nous passons au dynamique, à l'interactif. Le lecteur devient acteur en enfonçant des boutons du style magnétoscope, en activant des liens, etc.

La dimension temporelle

C'est à cette quatrième dimension, encore peu explorée jusqu'à présent en informatique, que s'intéresse ce mémoire : il traite des documents multimédia structurés, en particulier de leur *dimension temporelle*. L'auteur d'un document multimédia structuré devra spécifier non seulement le contenu et la structure de son document, mais également son scénario temporel. Il devra pouvoir ajouter des relations temporelles simples ("avant", "en même temps", "après", etc) entre les éléments constitutifs de son document. Ces relations indiquent des contraintes qu'il faudra respecter lors de la présentation du document. Elles forment un système de contraintes temporelles qui est au cœur du problème.

Le sujet du mémoire

La gestion des contraintes temporelles soulève de nouveaux problèmes, dont certains ne sont toujours pas résolus. C'est exactement de cela dont traite ce mémoire. C'est également cela qui a servi de cadre théorique pour mon stage à l'INRIA de Grenoble. Parmi les problèmes à résoudre figurent la vérification de la cohérence du système de contraintes temporelles et la recherche d'une de ses solutions.

A cette fin, un algorithme a été mis au point : l'algorithme classique *PC2* (pour Path Consistency II). Par une technique de propagation de contraintes, il nous permet de vérifier la cohérence d'un système et de calculer les domaines minimaux associés. C'est cet algorithme

en particulier que j'ai implémenté et étendu aux documents hiérarchiques durant mon stage. La particularité du présent mémoire tient dans les algorithmes cités aini que dans le chapitre 8. Ce dernier ajoute aux algorithmes une nouvelle extension qui devrait nous permettre de gérer les intervalles incontrôlables des systèmes de contraintes temporelles (partiellement du moins).

1.2 Plan de ce mémoire

Maintenant que nous savons exactement ce que le présent mémoire va développer, nous allons en présenter la structure.

Nous traitons de l'édition de documents multimédia structurés. Il est donc normal de s'intéresser aux aspects touchant à l'édition de documents multimédia, dans le chapitre 2. Après un rapide tour d'horizon des divers approches actuelles, nous tentons de les comparer. Parmi elles, les approches par contraintes semblent mieux satisfaire les besoins des auteurs.

L'un des logiciels utilisant cette approche intéressante est Madison et son langage Madeus. Le chapitre 3 est consacré à ce logiciel informatique (écrit en C, puis en Java). Madison sert à l'édition et à la présentation de documents multimédia structurés. L'approche retenue est qualifiée de "déclarative" ou de "relations temporelles spécifiées par contraintes". Après avoir spécifié la structure et le contenu de son document, l'auteur définit des relations spatio-temporelles entre les éléments constitutifs du document. Ces relations forment deux systèmes de contraintes indépendants.

Les systèmes de contraintes utilisés font partie d'une sous-classe de problèmes de satisfaction de contraintes temporelles (TCSP), appelée STP, pour Simple Temporal Problem. Les concepts nécessaires pour comprendre mieux ce que sont ces contraintes sont présentés dans le chapitre 4.

Ces systèmes de contraintes temporelles soulèvent des problèmes abordés dans le chapitre 5. Par exemple, un tel système peut ne pas admettre de solution. Il faut donc pouvoir détecter les spécifications impossibles, incohérentes. Ce chapitre est consacré à la définition des types de cohérences, ainsi qu'à leur détection automatique.

Il existe un algorithme détectant les incohérences et calculant de plus ce que l'on appelle les graphes minimaux, si le document est cohérent. C'est l'algorithme classique PC2, dont les principes sont décrits dans le chapitre 6.

Les théories étant fixées, il faut les confronter à la réalité de l'implémentation. Plusieurs problèmes laissés dans l'ombre sont révélés dans le chapitre 7. Il traite de mon implémentation de PC2 à l'INRIA. Il comporte également des extensions de l'algorithme PC2, comme celle destinée à vérifier la cohérence de documents hiérarchiques. Cet algorithme est utilisé aussi pour le formatage des documents. Le plus gros problème rencontré est celui des performances en temps d'exécution. Ce chapitre se termine donc par quelques éventuelles pistes à suivre pour fournir une amélioration de ces performances.

Les chapitres précédents traitent des cas simples, où tout est déterminé statiquement, à l'avance. Mais cela n'est pas suffisant en pratique. Il existe un type d'intervalles temporels ne pouvant être présentés qu'après des calculs dynamiques, pendant la présentation même du document. Le chapitre 8 nous confronte à ce type d'intervalles, qualifiés d'incontrôlables. Ils génèrent des problèmes complexes, étant donné leur dimension dynamique. Il s'agit d'un problème crucial pour la suite du développement de Madeus. C'est pourquoi nous nous y arrêtons. Après avoir expliqué le problème, nous tenterons de fournir des pistes pour résoudre

partiellement ce problème épineux.

Le chapitre 9 présente la traditionnelle section des perspectives à venir et autres conclusions.

Enfin, ce mémoire présente des informations annexes : une table des matières et une liste de figure au début, ainsi que trois annexes, la bibliographie et un index à la fin. Les annexes concernent la comparaison de logiciels d'édition multimédia, la DTD du langage Madeus, et un aperçu des expériences que j'ai acquises au cours du stage et de la rédaction de ce mémoire.

Voilà pour le plan de ce mémoire. Nous ajoutons que ce mémoire est lui-même un document structuré : il a été rédigé en LaTeX.

Tout commentaire est évidemment le bienvenu[3]. En attendant, nous vous souhaitons d'ores et déjà une agréable lecture.

3. Comme par exemple le manque flagrant d'humour tout au long du document ;-)

Chapitre 2

Edition temporelle de documents multimédia

2.1 Plan

Comme nous l'avons dit, nous nous intéressons aux documents multimédia. Il est donc logique de se pencher sur les éditeurs de tels documents. Voici comment nous allons procéder. Nous commençons par définir le terme multimédia (section 2.2), pour mieux situer le sujet. Ensuite, nous identifions une série de critères pour classifier les nombreuses méthodes d'édition (section 2.3.1 page 19). Avec eux, nous allons comparer les méthodes d'édition multimédia. Nous mettrons en évidence deux types d'approches : les approches opérationnelles (section 2.4 page 22) et les approches déclaratives, connues également sous le nom d'approches relationnelles à base de contraintes (section 2.5 page 28). Nous examinerons les avantages et inconvénients de chaque paradigme appartenant à chacune de ces deux approches. Nous en établirons une synthèse (section 2.6 page 32). Puis nous aborderons les standards existants, car ils sont nécessaires à l'échange de documents multimédia (section 2.7 page 34).

2.2 Multimédia : quid ?

Commençons par définir le terme "multimédia". Le multimédia est à l'honneur, paraît-il. Mais qu'entendons-nous exactement par "multimédia" ? C'est ce que nous découvrons dans cette section.

Il y a quelques années, ce terme était utilisé à tort et à travers par des vendeurs de matériel informatique peu scrupuleux, dans le but (lucratif) d'augmenter leur volume de vente. Tentons donc de démystifier ce terme magique. Le petit Larousse (édition 1991) nous apporte un début de réponse :

> **Multimédia**. adj. Qui utilise ou concerne *plusieurs médias*. (...)

Cela correspond à une définition littérale du terme. Un document multimédia est donc un document comportant plusieurs (types de) médias : les textes, les images, les animations, le son et la vidéo sont des exemples classiques. Il en existe d'autres, plus exotiques, comme les pages web ou les applets. Ce sont, avec les interactions utilisateurs, les briques de base qui vont servir à élaborer un véritable document interactif : un document multimédia.

Cette définition n'est pas complète — elle date d'ailleurs seulement de 1991. Un site web spécialisé dans la définition de termes informatiques, Webopedia [Def99], nous apporte un complément d'information (en anglais) :

> **Multimedia**. The use of computers to present text, graphics, video, animation, and sound *in an integrated way*. (...) With increases in performance and decreases in price, however, multimedia is now commonplace. (...)

Le caractère additionnel d'intégration nous permet de ne pas qualifier de multimédia une application de PAO (Publication Assistée par Ordinateur). Une définition plus précise et rigoureuse a été donnée dans [Layaïda97, pg 9–10]. Nous y apprenons qu'il faut trois dimensions pour obtenir la qualification de multimédia :

- un nombre suffisant de médias manipulables
- une nature temporelle parmi certains de ces médias
- un haut niveau d'intégration au niveau du traitement de ces médias

Nous obtenons finalement une définition complète, tirée ce ce même ouvrage :

> Un système ou une application est qualifiée de **multimédia** si il ou elle supporte le traitement intégré du plusieurs médias dont au moins un est de nature temporisée.

2.3 Classification des approches d'édition temporelle

Nous avons enfin notre définition du terme "multimédia". Nous souhaitons pouvoir en éditer. Penchons-nous donc sur le domaine de l'édition multimédia.

La nouveauté apportée par les documents multimédia par rapport aux documents classiques consiste évidemment en leur *dimension temporelle*. Fini les documents figés : place aux animations, aux documents interactifs. Les présentations de ce type de documents sont bien plus attrayantes. Encore faut-il disposer d'outils pour les créer : comment bâtir un document multimédia à partir des quelques briques que sont ses médias constitutifs ?

La réponse doit être nuancée. Comme souvent en informatique, il existe bien plus d'une seule manière de faire.[1] Nous allons commencer par en décrire certaines. Dans le chapitre 3, nous détaillerons celle retenue par l'équipe *Opera* de l'INRIA. En attendant, procédons à un état de l'art dans le domaine de l'édition (et un peu de la présentation) de documents multimédia. Quelles sont donc ces différentes approches ?

Encore quasi inexistants il y a quelques années, les éditeurs multimédia se multiplient. Dans le domaine de l'édition multimédia, un site web [Faq99] vaut le détour. Il s'agit d'une "Foire Aux Questions" (FAQ) bien fournie. Des outils d'édition sont listés par plateforme, avec une petite description. Même si certains outils sont repris plusieurs fois selon la plateforme supportée, cette liste comporte presque 14 pages ! Ils sont en réalité bien plus nombreux : le parcours du site [Tools-1] suffira amplement à convaincre les sceptiques. Bref, les outils d'édition ne manquent plus.

1. *"There's More Than One Way To Do It"* (TMTOWTDI, prononcé "tim-toady") est d'ailleurs le slogan des programmeurs utilisant le langage *Perl*.

2.3.1 Critères

Les outils d'édition sont foison. Chaque logiciel a sa façon de faire, bien particulière. Comment s'y retrouver? Il faut disposer de critères pour les classer.

Une étude y a été consacrée [Jourdan98b]. Elle identifie deux groupes de critères. Le premier concerne l'expressivité de langage définissant un document et le second concerne les besoins en terme d'édition. Ce sont ces critères, numérotés, que nous présentons ci-dessous.

Expressivité du langage

L'expressivité du langage se divise en trois sous-groupes de critères. D'abord, ceux liés au caractère intrinsèque des objets multimédia :

- (1) *Large variété d'objets de base.* Les textes, la vidéo, l'audio, les images fixes, les animations, les applets, par exemple. On distingue deux catégories : les objets *discrets*, dont le rendu est immédiat (texte, image fixe) et les objets *continus*, dont le rendu est réparti en intervalles de temps réguliers (audio, vidéo). Nuançons tout de même : il est vrai que ce critère n'est pas vraiment un critère d'expressivité du langage considéré. Il s'agit plutôt d'une mesure des capacités de l'éditeur à manipuler un nombre suffisant de médias pour intéresser les auteurs de documents. C'est un aspect très minime du langage, en tout cas pas très révélateur.
- (2) *Contrôle des objets continus.* En cas de synchronisation entre objets, l'auteur doit pouvoir déterminer si le contenu d'une vidéo doit être présenté en entier ou bien s'il peut être tronqué.
- (3) *Capacité d'interaction.* Il faut pouvoir rendre activables certains objets pendant une période de temps donnée. C'est le cas des objets interactifs comme les boutons ou les liens hypertexte. Notons qu'une image peut très bien devenir un bouton.
- (4) *Définition de styles temporels.* Il existe deux sortes de styles temporels. Les premiers sont ceux du genre PowerPoint, comme la diminution de volume d'un audio, ou la disparition progressive d'une vidéo à l'écran. Il est intéressant pour l'auteur de pouvoir activer et désactiver des styles associables aux objets. Mais cela ne relève pas vraiment de l'expressivité du langage. Les seconds styles sont en fait une composition des premiers : ce sont ceux qui doivent être gérés entre plusieurs objets. Un exemple est la disparition progressive d'une vidéo se déroulant en même temps que la diminution du volume d'un commentaire audio l'accompagnant. La capacité de pouvoir formater ce second type de styles temporels constitue le critère numéro 4.
- (5) *Support des objets non prédictibles (contrôlabilité).* Les objets non prédictibles sont de deux types. Les premiers sont ceux dont la durée ne peut être prévue statiquement (charge système pour une vidéo, accès réseau pour une image, etc). Les seconds sont les éléments interactifs. Sachant que nous pouvons modifier dynamiquement la qualité d'une vidéo pour en garantir la durée dans un système surchargé, il nous faut une définition précise du caractère prédictible ou non :

  > Un objet est considéré comme *prédictible* ou *contrôlable* dans un environnement donné si et seulement si le système de présentation garantit sa durée sans perte de sens.

Ensuite, voici un critère concernant la composition temporelle

- (6) *Composition temporelle.* Les objets doivent pouvoir être combinés dans le temps de manière arbitrairement complexe. Le pouvoir d'expression du langage temporel (expressivité) est souvent comparé avec le langage idéal des compositions d'intervalles, celui des relations d'Allen [Allen91]. De plus, l'auteur devrait pouvoir spécifier les relations temporelles entre les objets sans se soucier de répercussions dues à leur ordre d'insertion.

Enfin, un critère d'interactivité, relatif notamment à la navigation hypermédia.

- (7) *Interactivité.* L'interactivité d'un document est de plusieurs ordres. Le premier concerne la navigation et le second les autres aspects.
  - D'abord, la navigation à l'intérieur et entre documents. Il existe deux sortes de telles fonctions de navigation. D'une part, il y a les fonctions style magnétoscope (pause, stop, resume, etc.) Elles sont indépendantes du document et de sa spécification. Elles ne nous intéressent donc pas. D'autre part, il y a les interactions propres à chaque exécution d'un document. C'est le cas des boutons à enfoncer pour déclencher des actions. C'est également le cas des liens hypertexte, utilisés pour naviguer aussi bien spatialement que temporellement dans les documents. Dans la dimension temporelle, on peut citer les "pause", "stop", "resume", "aller en seconde partie du scénario", "sauter au chapitre 4", etc.
  - Ensuite, il reste tous les innombrables aspects liés à l'interactivité. Cela regroupe énormément de possibilités. Par exemple, le remplissage de champs d'un formulaire, à la manière du Web[2]. Cela peut vouloir dire encore beaucoup plus. Ainsi, il est possible qu'un document soit un véritable jeu vidéo. Il existe un tel exemple de jeu sur Internet. Il s'agit d'un document écrit avec Macromédia et son langage de script, Lingo. Le jeu est inspiré de la série "South Park" qui déchaîne les passions outre Atlantique. Le but est le suivant. De gentils enfants se trouvent de l'autre côté d'une rivière. Du bon côté, de méchants bulldozers traversent rapidement l'écran de gauche à droite. Le lecteur du document (devrait-on dire l'acteur?) doit aller chercher les enfants de l'autre côté pour leur faire traverser la rivière en toute sécurité, en évitant les méchants. Il est évident que les éditeurs utilisant la programmation (souvent des scripts) sont imbattables sur ce plan.

Capacités d'édition

Cette nouvelle classe de critères concerne la rapidité de la mise sur pied d'un scénario :

- (8) *Facilité d'utilisation.* Il faut viser un public d'auteurs le plus large possible. Il ne faudrait pas par exemple qu'une habilité à la programmation soit requise pour élaborer correctement un scénario, même complexe.
- (9) *Création directe.* La spécification du document que l'auteur a en tête doit être aisément transposable via l'éditeur. Il ne devrait pas se soucier de pertes d'informations dues à la traduction de son scénario dans les concepts utilisés par son éditeur. De plus, il devrait pouvoir spécifier des objets et leurs relations sans tenir compte de l'ordre d'insertion de ces relations.

2. Le fameux "World Wild Web", plus connu sous le nom de "Web sauvage" ;-)

- (10) *Gestion des scénarios indéterministes.* Les éléments non prédictibles génèrent parfois plusieurs présentations possibles d'un même document. Il faut fournir à l'auteur un moyen de visualiser ces possibilités et de naviguer entre elles. Pour cela, il faut mettre sur pied des outils de visualisation.
- (11) *Modifications locales aisées.* Ce critère et le suivant font appel au caractère **incrémental** de l'édition : le résultat final n'est que très rarement atteint dès la première spécification, car nous sommes en face d'un cycle classique "spécifier, tester et modifier". Les modifications locales peuvent entraîner des modifications en chaîne, du point de vue structurel et spatial. L'éditeur devrait les prendre en charge, car elles sont sujettes à erreur humaine.
- (12) *Rapidité de passage entre édition et présentation.* Ce passage est inévitable, à cause de la nature dynamique du document. Ne pouvant appliquer les principes classiques du Wysiwyg[3], il faut permettre un passage rapide entre ces deux modes.
- (13) *Capacités d'abstraction.* Une notion de hiérarchie permet de gérer de gros documents, ainsi que la réutilisation de ses parties.
- (14) *Modèles de documents multimédia.* Dans le but d'accélérer le processus, il faudrait disposer de modèles, comme le permettent les éditeurs de documents statiques avec leurs hordes d'assistants automatiques.
- (15) *Support multi-grilles.* Il faut pouvoir adapter un document en fonction de son lecteur (niveau, langue, etc). Ceci sous-entend la réutilisation d'éléments multimédia communs.

2.3.2 Classification

Armés de tous ces critères, nous pourvons classer la masse disponible des logiciels d'édition multimédia et les comparer entre eux. Nous choisissons une classification en deux groupes distincts, adaptés chacun à un style particulier d'édition. Cette distinction est de la même nature que celle qui différencie les approches impératives des approches fonctionnelles ou logiques, en programmation :

1. Les éditeurs à approche **opérationnelle**. L'auteur spécifie exactement comment le scénario doit être exécuté, via un langage de script ou une structure opérationnelle (arbre ou réseau de Pétri). La phase de présentation implémente directement la sémantique de la structure opérationnelle utilisée. Dans cette approche, l'auteur doit souvent donner plus d'informations temporelles.
2. Les éditeurs à approche **déclarative / par contraintes**. L'auteur spécifie quel scénario il veut (de manière déclarative), sans se soucier de comment il sera obtenu (actions opérationnelles). Tout repose sur la programmation par *contraintes*. La phase de présentation commence par transformer automatiquement la déclaration en structure opérationnelle : c'est la phase de *formatage*. Idéalement très courte, cette phase additionnelle calcule les instants de départ et les durées de chaque élément à présenter. C'est le même style de principes qu'applique le formateur de documents LaTeX.

Rappelons qu'une classification n'est jamais parfaite, car les critères utilisés, rarement orthogonaux, ne permettent pas toujours de répartir univoquement toutes les situations possibles. De plus, il en existe parfois d'autres. La foire aux questions à propos des systèmes d'édition

3. What you see is what you get

multimédia [Faq99] en propose d'ailleurs une autre. Mais il n'est pas très difficile de ranger leurs catégories dans les nôtres.

2.4 Approches opérationnelles

Une première classe d'éditeurs est rassemblée sous le terme d'approches opérationnelles. Ce sont les plus fréquement utilisées, les plus commerciales. Mais sont-elles pour autant les meilleures? Cette première classe d'éditeurs peut être elle-même divisée en sous-classes, en paradigmes, au nombre de 4 (timeline absolues, scripts, arbres, réseaux de Pétri).

Dans les approches opérationnelles, l'auteur spécifie les durées exactes de son scénario. Au risque de paraître trop partiaux, relevons-en déjà les limites. Supposons que l'auteur veuille voir une image A et une vidéo B en séquence, le tout en même temps qu'un son C. C'est le schéma de la figure 2.1 (page 22). L'auteur doit faire un choix. Le premier est de fixer les durées de A, B et C. Mais ce scénario sera difficile à modifier par la suite, s'il change la durée de A, par exemple. Le second choix est de demander que le plus court entre B et C coupe l'autre, mais le contenu de l'un des deux ne sera pas pas présenté en entier au lecteur. Remarquons que dans chacun des choix possibles pour ce scénario pourtant très simple, l'auteur risque de ne pas être satisfait.

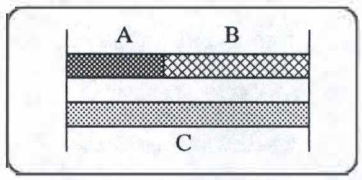


FIG. 2.1 – *Approche opérationnelle : limites pour un scénario simple*

L'approche opérationnelle rassemble quatre familles, quatre paradigmes, selon la manière de gérer l'édition. Chaque catégorie ayant ses inconvénients, beaucoup utilisent des approches combinées. Dans ce cas, nous retenons la plus importante. La classification ci-dessous est inspirée d'une analyse de [Layaïda97, pg 32–40] et [Jourdan98b]. Ces familles d'éditeurs sont fondés sur :

- les *timeline absolues* (axes du temps absolus). L'utilisateur positionne chaque élément de son document de manière absolue par rapport à une ligne du temps. (section 2.4.1)
- les *scripts* (la programmation). Pour obtenir des résultats très fins, les auteurs spécifient la synchronisation du document via un langage de programmation (généralement orienté objet). Le document, représenté par le déroulement de son scénario, son flot de contrôle, peut même être figuré graphiquement, à la manière des organigrammes. (section 2.4.2 page 24)
- Les *arbres* (hiérarchies). Cette technique connue peut être utilisée pour représenter (partiellement) les synchronisations, à un niveau plus abstrait : les feuilles sont les objets, tandis que les nœuds sont les relations temporelles. (section 2.4.3 page 25)
- Les *réseaux de Pétri*, bien connus dans le domaine de la modélisation de processus temporels et de leurs synchronisations. (section 2.4.4 page 27)

Nous allons détailler chacun d'eux, tour à tour.

2.4.1 Édition via les lignes du temps absolues (timeline absolues)

Les lignes du temps absolues sont un premier paradigme opérationnel. Voici d'abord leur principe de fonctionnement. Ces éditeurs sont parmi les plus répandus dans les systèmes commerciaux, sans doute grâce à leur métaphore très simple. Leur interface présente une ligne du temps absolue comme référence. Elle comporte aussi des axes parallèles, qui permettent à l'auteur d'y placer ses médias, de manière éventuellement concurrente. C'est un système similaire à celui des pistes utilisées dans la spécification de documents sonores au format midi. De cette manière, le début de chaque média est précisé de façon absolue par rapport au début du document, ainsi que sa durée. Pour permettre un placement assez fin, les subdivisions de ces lignes du temps sont généralement très petites, de l'ordre d'une image d'une séquence vidéo. Les éditeurs plus développés offrent des fonctionnalités supplémentaires. Ainsi, un auteur peut associer à chaque média des propriétés (effets de style lors des transitions, etc)

Le succès de cette approche tient dans l'extrême simplicité pour l'auteur. Mais les possibilités d'édition en souffrent beaucoup. C'est pourquoi cette approche est presque toujours accompagnée d'une approche de scripts (puissants, mais trop complexes pour l'auteur moyen).

Avant de passer à la critique, il faut souligner que nous ne parlons ici que des timeline absolues. Il existe des systèmes timeline où le placement des objets peut être relatif. D'ailleurs, Madison utilise la métaphore de la ligne du temps dans sa vue "scénario". Mais dans ce dernier cas, les placements sont tous relatifs, et le modèle à la base utilise une autre représentation. Voici une analyse critique plus détaillée des timeline absolues.

Avantages

- *Simplicité d'utilisation.* C'est idéal pour l'auteur moyen (critère 8). La métaphore de la ligne du temps est très proche de l'univers cognitif du concepteur de document. Il ne doit pas faire un effort énorme pour mettre en œuvre avec ces éditeurs la représentation mentale qu'il s'est forgée de son futur document. Le succès commercial est sans doute justifié par ce caractère fort intuitif.

Inconvénients

Il existe des inconvénients, bien sûr. Ceux-ci serviront à justifier l'approche adoptée par le langage Madeus du logiciel Madison.

- *Incrémentalité.* Le positionnement des médias est absolu par rapport au début du document. Cela peut poser problème dès que l'on modifie la durée, la date de début ou de fin d'un média. Supposons que deux médias A et B soient consécutifs. Si nous modifions la durée de A, il faudra modifier l'instant de début de B. Ceci est susceptible d'engendrer des changements en cascade. L'utilisateur va devoir effectuer beaucoup de modifications lui-même. Il risque d'oublier de répercuter ces modifications sur certains éléments, ou d'effectuer des positionnements imprécis, voire incorrects. De plus, les tâches répétitives ou sujettes à erreur devraient être entièrement prises en charge par le système. Bref, mis à part le critère 8, les autres critères d'édition ne sont guère remplis. Ainsi, le critère 11(facilité des modifications locales) et le critère 9 (importance de l'ordre d'insertion des relations) sont écartés.
- *Structuration.* Ce système ne tient aucun compte de la structure des documents (critère 13). Les documents créés sont plats, et ingérables pour peu qu'ils soient suffisamment

étoffés. C'est tout le contraire des standards actuels comme HTML, XML ou SGML. Ceci rend impossible la réutilisation aisée de parties de documents, comme par exemple une introduction multimédia commune à tous les documents qu'une entreprise émet.

- *Incontrôlabilité.* L'auteur doit donner de manière absolue le début et la fin de chaque élément multimédia. Or, il existe des éléments dont la durée n'est pas prévisible. Les cas les plus représentatifs sont ceux des interactions utilisateur ou des médias téléchargés via un réseau. Le problème des interactions utilisateur peut être contourné via un langage de script, mais la convivialité en pâtit. De plus, comment gérer les présentations dont la durée ou la qualité doivent être modifiées dynamiquement à cause d'une charge réseau trop lourde? Bref, le critère 10 n'est pas respecté non plus.

Représentant

Le plus connu des éditeurs basé sur ce principe est sans conteste le logiciel *Director* [Tools-2] de la firme *Macromédia.* Les documents créés peuvent être visualisés sur le web grâce au plugin *shockwave* [Tools-3]. Pour pallier à certaines limites, Director inclut un langage de script orienté objet, appelé *Lingo.* Ce logiciel leader est utilisé pour produire des présentations multimédia à caractère professionnel.

2.4.2 Édition via la programmation (scripts)

Les scripts sont le second paradigme opérationnel. Les approches se servant d'illustrations graphiques au niveau de l'interface utilisateur pour la représentation temporelle privilégient la facilité pour l'auteur. Mais l'inconvénient habituel intervient : le pouvoir d'expression autorisé est bridé. Un utilisateur exigeant et expérimenté ne supportera pas ces limites. Cette situation est assez classique en informatique. Ainsi, nous pouvons choisir entre un système d'exploitation "*user-friendly*" à la Windows (qui empêche de contrôler vraiment sa machine), ou un système plus puissant à la Unix/Linux, mais dont la convivialité laisse à désirer pour les non initiés.

C'est ici qu'intervient le paradigme des scripts. Pour obtenir sur un scénario un contrôle plus fin ou des synchronisations très complexes, le seul moyen véritable est l'utilisation d'un langage de programmation (généralement orienté objet pour la facilité). Le pouvoir d'expression de relations spatio-temporelles à l'intérieur d'un document n'a alors plus vraiment de limites. De manière concrète, ces langages peuvent être de simples scripts[4], ou même des librairies de modules qui étendent une application existante.

Cette approche est souvent le complément d'autres, sans aucun doute. Il existe d'ailleurs beaucoup de langages de scripts développés spécialement pour l'édition multimédia. Un site éducatif du MCLI [Lang99], le *Maricopa Center for Learning and Instruction*, a d'ailleurs recensé 70 langages d'édition (authoring languages) !

Avantages

- *Expressivité.* L'expressivité des relations spatio-temporelles est maximale (expressivité, critères 1-7). Les utilisateurs expérimentés y trouveront certainement de quoi étancher leur soif. Les possibilités offertes sont pour ainsi dire quasi infinies.

4. Les scripts sont souvent préférés aux "véritables" langages de programmation impérative : ils ont peu de contraintes quant à la spécification de types de données. De plus, ils ont tendance à être plus simples et plus souples.

Inconvénients

Cette approche procédurale pose problème pour les critères d'édition (8–15).

- *Facilité d'utilisation.* La création d'un document requiert plus de compétence, et devient hors de portée pour un auteur inexpérimenté dans le domaine de la programmation (critère 8).
- *Réutilisation.* De plus, les synchronisations sont éparpillées et enfouies dans le code. En conséquence, la réutilisabilité des documents ainsi créés paraît plus qu'improbable. (critère 13). Cela rend difficile une vue globale du document, du moins sans l'exécuter.
- *Mise à jour.* Ne parlons pas de la difficulté de leur mise à jour (critères 9 et 11), semblable à la complexité de maintenance d'un programme.
- *Lenteur.* Enfin, si seul ce paradigme est utilisé, le temps de développement d'une présentation multimédia est multiplié par un facteur important (un facteur 8 est courant).

Représentant

A la limite, nous pourrions choisir n'importe quel langage de programmation comme représentant de cette approche (Java ou autres). Mais souvent, un autre paradigme sert de base, et un langage de script additionnel vient pallier aux limitations lorsque cela devient nécessaire. Un exemple connu est le langage orienté objet "Lingo", qui accompagne le célèbre Director. Un autre exemple est le standard MHEG, décrit plus bas, qui incorpore dans un modèle orienté objet des listes de propositions "événement et condition vérifiée ⇒ action".

Un autre exemple est plus frappant, car nous pourrions le classer erronément dans une autre catégorie. Il s'agit du logiciel *Icon Author* de Asymetrix [Tools-4], dont une capture d'écran se trouve à la figure 2.2. Le but étant de représenter un document par son flot de contrôle, il est possible de le faire graphiquement, à la manière des organigrammes représentant un petit programme. C'est l'approche adoptée dans ce logiciel. Son fonctionnement est très simple: l'auteur choisit une icône (gauche de l'écran). Ensuite, il la glisse sur le graphe et édite ses propriétés. En prenant l'analogie des organigrammes, nous pouvons voir une icône comme une commande, ainsi que le document comme un programme. Plutôt que d'écrire un script de commandes, l'auteur crée un graphe de contrôles visuels. Finalement, il s'agit d'offrir (une partie de) la puissance des scripts aux non-informaticiens, ce qui atténue l'inconvénient principal de cette approche.

2.4.3 Édition via des arbres hiérarchiques

Les arbres constituent le troisième paradigme opérationnel. Les structures d'arbre sont bien connues dans le domaine des documents structurés[5], pour représenter la décomposition hiérarchique d'un document. De nombreux algorithmes y sont associés. Une fois ces arbres adaptés au domaine multimédia, la dimension temporelle peut être spécifiée comme suit: les objets de base sont représentés par les feuilles de l'arbre, tandis que les opérateurs temporels sont figurés par les nœuds internes de l'arbre.

5. Il existe des langages et métalangages standardisés de description de documents, comme LaTeX, HTML, XML ou SGML

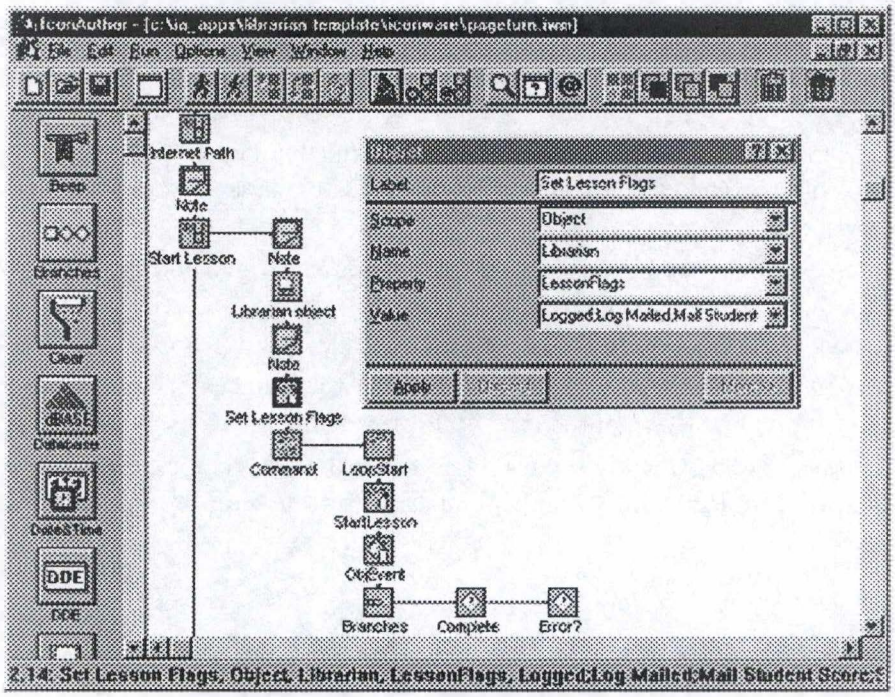


FIG. 2.2 – *Approche par scripts avec métaphore de graphe : IconAuthor*

Représentants

Un premier est CMIFed [Acm93] — ses opérateurs temporels sont les opérateurs séquentiels et parallèles. Un second est le langage SMIL [Smil-1] — ses opérateurs, plus nombreux, incluent "A par_min B" (A et B commencent en même temps, et le plus court des deux termine l'autre). Notons au passage que l'application développée à l'INRIA, Madison, supporte nativement le langage local (Madeus), mais permet aussi de lire des documents au format SMIL.

Avantages

- *Expressivité.* Le niveau de spécification est plus général et offre une meilleure perception de la structure temporelle du document.
- *Lectures multi-grilles.* Dans le cas de l'environnement CMIFed, le critère 15 est respecté, car il comporte une vue "Canaux", présentant chaque type de média sur des lignes séparées, parallèles à un axe de temps absolu.

Inconvénients

- *Pouvoir d'expression.* Celui-ci est limité, car certains scénarios ne peuvent être exprimés (critères 5 et 6). Ainsi, supposons qu'un lecteur démarre une animation en appuyant sur un bouton. Supposons aussi que la fin de cette animation doit être synchronisée avec la fin d'un autre élément. Cela exige de calculer dynamiquement la durée ou la vitesse de l'animation, à l'exécution.

- *Création directe.* Le critère 9 n'est pas respecté, car les spécifications ne sont pas toutes immédiates. Supposons que nous ayons en parallèle une séquence A, B avec une séquence X, Y et que l'objet X doive démarrer t secondes avant l'objet B. Pour plus de clarté, consultons la figure 2.3. La seule manière d'obtenir ce scénario est d'ajouter un délai D avant l'objet X, en fixant précisément sa durée. Mais alors, la relation de précédence entre X et B n'est pas représentée directement.

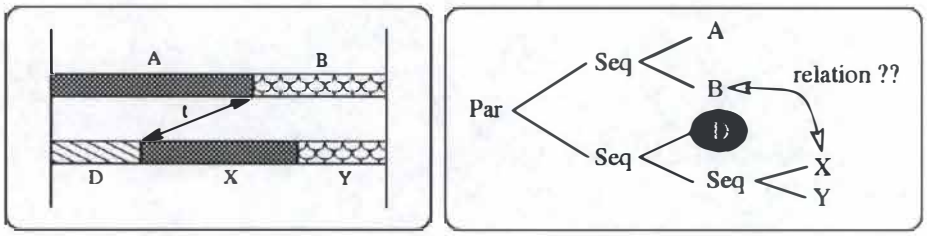


FIG. 2.3 – *Approche par arbres. Représentation non immédiate d'un scénario avec relation "X before(t) B"*

- *Modifications locales.* Elle ne sont pas aisées (critère 11). Les durées des objets ne sont pas recalculées automatiquement en cas de changement du scénario. Dans l'exemple précédent, la modification de la durée de l'objet A n'est pas répercutée automatiquement.
- *Note.* Pour pallier à ces deux problèmes, CMIFed a ajouté la notion d'arcs de synchronisation. Mais, outre le fait que la sémantique en est peu claire, cela pose des problèmes de cohérence avec la structure existante. Il est même possible de générer des deadlocks, qu'il faudrait pouvoir détecter. Cela ressemble à un petit pansement : utile, mais guère efficace.

2.4.4 Édition via les réseaux de Pétri

Les réseaux de Pétri sont le quatrième et dernier paradigme opérationnel. Ils sont bien connus dans le domaine de la modélisation de processus parallèles, pour exprimer des synchronisations et effectuer des tests concernant certaines propriétés (safety, liveness). Ces réseaux ont été utilisés dans le domaine de l'édition multimédia. Les objets sont modélisés par des places, et l'information temporelle est associée soit aux places (durées dans OCPN), soit aux arcs (intervalles de validité dans HTSPN). Les transitions sont étiquetées par des opérateurs temporels (séquence, parallélisme dans OCPN, ou un ensemble plus riche avec par_min, etc dans HTSPN). Dans le cas de HTSPN, une organisation hiérarchique est représentable : il suffit que des places abstraites représentent un sous-réseau.

Représentants

Deux d'entre eux viennent d'être cités : OCPN et HTSPN.

Avantages

Les réseaux de Pétri sont plus adaptés que les arbres hiérarchiques pour spécifier la dimension temporelle d'un scénario :

- *Création directe.* Contrairement aux inconvénients de l'exemple précédent (figure 2.3 page 27 dans l'approche par arbres), le lien de précédence entre C et B n'est pas perdu.

En effet, comme nous pouvons le constater sur la figure 2.4, la relation "X before(t) B" est directement représentable.

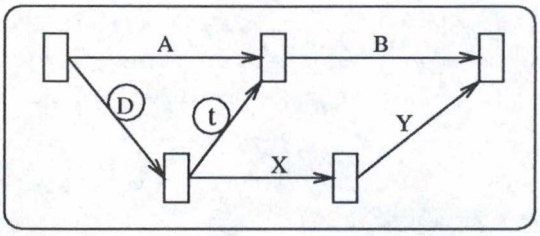


FIG. 2.4 – *Approche par réseaux de pétri : représentation immédiate d'un scénario avec relation "X before(t) B"*

Inconvénients

- *Création directe.* La création de la structure n'est pas directe. Même si l'information de précédence n'est pas perdue dans l'exemple précédent, il faut néanmoins ajouter manuellement un délai D, comme nous le constatons sur la figure 2.4. (critère 9)
- *Modification locale.* Par contre, la modification de la durée de l'objet A dans ce même scénario devra être répercutée par l'auteur sur le délai D, sans aide automatique. (critère 11)
- *Incontrôlabilité.* Les intervalles incontrôlables ne sont pas envisagé dans cette approche. (critère 5)
- *Facilité d'utilisation.* C'est trop difficile pour l'auteur. Outre le fait de devoir ajouter manuellement certains intervalles, il faut bien constater que la création d'une spécification sous forme de réseau de Pétri (arcs et places) n'est pas du tout du niveau de l'utilisateur moyen. Ceci reste vrai malgré la nature graphique de ce paradigme. (critère 8)

2.5 Approches déclaratives ou par contraintes

Nous venons de voir la première classe d'approches d'édition multimédia. Nous avons dit qu'il y en avait une deuxième, sans doute plus intéressante pour l'auteur. Ce sont les approches déclaratives, celles où l'auteur utilise des relations temporelles spécifiées sous forme de contraintes.

Nous commençons par les cerner quelque peu, en présentant quelques généralités, avantages et inconvénients. Au passage, nous préciserons la sémantique des triplets $[m, \lambda, M]$. Puis nous préciserons notre analyse, en détaillant un peu plus trois représentants de cette classe d'approche.

2.5.1 Généralités

Commençons par des considérations d'ordre général concernant les approches par contraintes. Nous en profitons pour signaler leurs forces et leurs faiblesses.

Le point de départ de cette classe d'approches est commun : il s'agit de la théorie des contraintes. L'auteur peut déclarer des relations d'instants ("=",">","<") ou bien des relations d'intervalles (comme les 13 relations de l'algèbre d'Allen [Allen91]).

Ces approches comportent toutes une phase supplémentaire de *formatage*, c'est à dire de transformation de la spécification en structure opérationnelle exécutable. Il faut pouvoir tout d'abord vérifier la *cohérence* temporelle: existe-t-il une solution, ou bien l'auteur a-t-il émis un scénario impossible? Ensuite, il faut calculer une solution, éventuellement optimale, qui vérifie toutes les contraintes. Idéalement, il faut aussi pouvoir naviguer entre les différentes solutions possibles.

Du côté programmation, deux classes d'algorithmes sont utilisées : la satisfaction de contraintes temporelles et la programmation linéaire.

Ces approches emploient un modèle du temps souvent le même: le modèle du temps élastique. Dans celui-ci, chaque objet est représenté par un intervalle auquel est associé un triplet $[m, \lambda, M]$ de durées: une durée minimale (m), une durée maximale (M), et une durée préférée (λ). Mais la sémantique exacte est plus nuancée et fait l'objet de la section qui suit.

2.5.2 Sémantique du triplet $[m, \lambda, M]$ de durée

Selon la nature des objets, la sémantique du triplet $[m, \lambda, M]$ est légèrement différente. Les objets peuvent être de 4 types différents, selon leur caractère contrôlable ou non et selon leur caractère discret ou continu. Un élément *discret* est un élément dont la restitution se fait en un instant, comme le texte et les images fixes. Un élément *continu* est un élément dont le rendu se fait lors d'intervalles régulièrement espacés dans le temps, comme la vidéo ou le son. Le caractère contrôlable ou non a été défini en page 19 dans la définition du critère numéro 5 de la section 2.3.1. Pour rappel, un élément est dit contrôlable si le système de présentation garantit sa durée sans perte de sens dans un environnement donné. Voyons donc la sémantique du fameux triplet selon le type de média.

Pour les objets contrôlables discrets (texte, image), il s'agit de durées choisies arbitrairement par l'auteur. Pour les objets contrôlables continus (vidéo, audio), il s'agit plutôt de durées acceptables liées à la qualité de la présentation. Nous pourrions considérer que l'auteur doit disposer d'une marge de manœuvre moindre pour fixer ce triplet, car une vidéo devant durer 6 secondes ne sera sans doute pas belle à voir si l'auteur lui demande de durer 3 minutes! Cependant, la situation n'est pas si simple: un auteur peut très bien vouloir présenter une partie de vidéo au ralenti, pour mettre un aspect en évidence. Dans le cas des objets contrôlables, le système d'édition peut donc choisir arbitrairement la durée d'un média à l'intérieur de l'intervalle que le triplet représente. Quant aux objets non contrôlables, il faudra trouver une définition adaptée à leur nature particulière, qui génère des problèmes complexes.

Nous avons donné brièvement les caractéristiques propres aux approches par contraintes. Avant de détailler plus avant 3 d'entre elles (Isis, Firefly et Madeus), voyons rapidement les avantages et inconvénients généralement associés à ces approches.

2.5.3 Avantages généraux

Les avantages que fournissent généralement les approches par contraintes sont les suivants :

- *Simplicité pour l'auteur.* Ces relations sont simples, intuitives (critère 8).
- *Composition temporelle.* L'ordre d'insertion des relations n'importe pas, et l'expressivité du langage est grande en utilisant les 13 relations d'Allen (critère 6).
- *Modifications locales.* Les modifications sont aisées, car il suffit d'ajouter ou retirer des éléments ou des relations. De plus, les durées peuvent être recalculées automatiquement

en cas de modifications locales. Ceci est particulièrement adapté à la nature incrémentale d'un processus d'édition. (critère 11).

- *Réutilisation.* La réutilisation est facilitée. Ainsi, si la spécification est suffisamment large, il ne faudra pas la changer en cas de traduction du document dans une autre langue. En effet, même si les durées des sons changent, il est probable que le système trouve encore une solution, si l'auteur n'a pas donné de durées trop strictes (triplet réduit à une valeur).

2.5.4 Inconvénients généraux

Les approches par contraintes génèrent souvent les mêmes inconvénients:

- *Rapidité.* Les phases additionnelles de vérification et formatage ne doivent pas être trop lentes. Il s'agit de l'une des principales difficultés rencontrées.
- *Interactivité.* Il n'est pas évident de prendre en compte l'interactivité dans ce modèle. Il génère très vite des objets incontrôlables. C'est également toute la problématique des opérateurs temporels d'interruption (par_min, etc). Ils sont nécessaires pour exprimer directement un scénario (critère 6). Notons que pour pouvoir contrôler les objets continus (critère 2), il faut pouvoir différencier l'opérateur "égal" (début et fin aux mêmes instants) de l'opérateur d'interruption comme "par_min" (début simultané et le plus court termine l'autre, en le tronquant éventuellement).
- *Incontrôlabilité.* Les objets de nature incontrôlable pose un problème intrinsèque complexe: leur durée n'est connue qu'à l'exécution. Le formatage doit devenir dynamique. De plus, une vérification dynamique de cohérence est loin d'être un problème trivial: des recherches en ce domaine se poursuivent actuellement.

Nous avons suffisamment présenté les approches relationnelles (par contraintes) dans leur généralité. Nous allons maintenant poursuivre en détaillant trois de leurs représentants, à savoir: Isis, Firefly et Madeus.

2.5.5 Edition avec Isis

Voici un premier paradigme appartenant aux approches par contraintes. Il s'agit d'un outil d'édition temporelle nommé Isis [Tools-5] Dans Isis, les relations temporelles utilisables sont les quatre relations de base d'Allen (meets, equals, starts, finishes). Les autres sont obtenues en ajoutant les délais adéquats. Toutes les durées sont considérées comme prédictibles (le critère 5 n'est pas respecté). Apparemment, les objets incontrôlables et les relations de causalité (interruption) ne sont pas pris en compte. Du côté interface, il existe une vue graphique qui montre plusieurs solutions: la flexibilité du scénario est représentée par des ressorts à l'intérieur des objets ayant un triplet de durées non réduit à une seule valeur.

Les algorithmes utilisés (programmation linéaire) optimisent une répartition équitable du temps entre les objets. Ensuite, pour des raisons de performances dans un environnement incrémental, le projet est passé aux réseaux de contraintes temporelles.

Dernièrement, le projet a opté pour une fonction d'abstraction manquante. Pour passer à cette capacité de structuration, les auteurs sont passés à un réseau de Pétri temporisé, où les places sont des objets composites décrits par leurs relations et où chaque arc décrit une interaction de l'utilisateur. Mais les auteurs eux-mêmes pensent que ceci entre en contradiction avec les avantages emportés par une approche à base de contraintes (critères 8 et 9).

2.5.6 Edition avec Firefly

Firefly est un second paradigme appartenant à la classe des approches par contraintes. Cette fois, ce sont les relations d'instants qui sont utilisées, et pas les relations d'intervalles d'Allen. Les synchronisations sont donc toutes celles concernant les instants de début et de fin de chaque objet multimédia. Ceci est représenté par une structure de graphe : les sommets sont des instants, tandis que les arcs sont des objets multimédia ou bien des relations d'instants accompagnées de l'habituel triplet de durées.

C'est le premier travail qui aborde le problème des intervalles incontrôlables (impliqué par certains médias). Ceci est une des directions de la recherche actuelle dans le domaine. Nous aurons l'occasion d'y revenir au chapitre 8. Pour assurer un formatage statique, le scheduler sépare le graphe d'instants en parties contrôlables et incontrôlables. Après avoir trouvé une solution pour chaque partie, un scheduler dynamique basé sur des événements gère l'intégration des parties.

L'algorithme utilisé pour trouver des solutions (optimales) est l'algorithme du simplex. Mais les performances en temps d'exécution sont trop faibles pour un environnement interactif et incrémental. (critère 12). De plus, Firefly ne supporte pas de possibilités d'abstraction du document (critère 13).

2.5.7 Edition avec Madison et Madeus

Voici un troisième et dernier paradigme appartenant à la classe des approches par contraintes. Madeus est le langage utilisé dans le logiciel Madison, développé à l'INRIA. Il fera l'objet d'une attention particulière au chapitre suivant. Cependant, nous allons dresser ici quelques caractéristiques importantes, permettant de le comparer aux autres approches. Ce sont les aspects hiérarchiques, les relations spécifiables et la gestion des intervalles indéterministes.

Hiérarchie / Abstraction

Un premier aspect distinguant Madison d'autres approches est la capacité d'abstraction, de description de documents hiérarchiques.

Contrairement aux deux approches précédentes, Madeus comprend nativement un support pour l'abstraction du document (critère 13). Il utilise l'organisation hiérarchique du document. Un document est composé d'éléments de base et d'éléments composés. Ces derniers servent à grouper un ou plusieurs éléments de base (ainsi que d'autres éléments composés) dans une entité plus large (représentation sous forme d'arbre).

Grâce à l'abstraction du document (hiérarchie), l'auteur peut facilement créer des documents complexes, à grande taille (critère 13). Les aspects multivues (critère 15) sont étendus à une visualisation hiérarchique parlante pour l'auteur[6]. De plus, cela facilite la réutilisation de parties de documents multimédia pour en réaliser un autre (introduction commune, etc).

Ceci avantage l'approche adoptée dans le langage Madeus. De fait, sachant que chaque élément comporte au moins les événements de début et de fin (ou même plus dans le cas de synchronisation avec un instant précis à l'intérieur d'une vidéo par exemple), nous devinons qu'une représentation événementielle de bas niveau va poser problème. En effet, dans un document important sans hiérarchie, la complexité de la description sera trop importante et vraiment illisible, surtout dans une représentation graphique.

6. Rappelons-nous la fameuse limite de 7 ± 2 attribuée à notre mémoire à court terme.

Les synchronisations sont définies uniquement entre éléments d'un même composite (leur père commun). Nous pouvons bien sûr spécifier une relation entre un objet de base et un composite, ce qui est pratique quand nous considérons le document à un plus haut degré d'abstraction. S'il faut percevoir une limite, c'est le fait qu'il n'est pas permis dans ce modèle de spécifier une contrainte temporelle entre deux éléments de niveaux hiérarchiques différents. Notons cependant que la visualisation de telles contraintes serait compliquée à réaliser. De plus, le fait d'associer un opérateur temporel à chaque nœud pose des problèmes qui sont évités de cette manière. Mais il faut admettre qu'un désavantage lié aux structures d'arbre se présente encore : certaines modifications (modifiant la structure du document) ne seront pas aisées (critère 11).

Relations temporelles

Un second aspect distinguant Madison des autres éditeurs est le nombre et le type des relations temporelles utilisables.

Dans Madison, celles-ci sont une extension des relations d'Allen : elles sont quantifiées. A cet ensemble s'ajoutent les relations de causalité "A Par_min B" (le plus petit des deux termine l'autre), "A Par_max B" (le plus rapide attends l'autre) et "A Par_master B" (quand A se termine, il tronque B s'il n'était pas déjà terminé). L'auteur spécifie ces relations de manière simple, via une palette les présentant de manière graphique (critère 8).

Indéterminisme

Le dernier aspect distinguant Madison est la gestion de l'indéterminisme, des intervalles incontrôlables, même si cela n'est pas encore implémenté actuellement.

Madeus permet l'utilisation d'éléments contrôlables et/ou incontrôlables. Les algorithmes utilisés pour la cohérence et le formatage utilisent des réseaux de contraintes temporelles. En effet, les techniques de propagation de contraintes sont bien adaptées à un environnement incrémental, contrairement à l'algorithme du simplex (critères 11 et 12). Ces algorithmes ont déjà été adaptés aux niveaux hiérarchiques. Par contre, les opérateurs d'interruption et les intervalles incontrôlables posent des problèmes complexes [Fargier98]. Il existe cependant un article récent traitant de l'intégration des opérateurs d'interruption dans un contexte de réseau de contraintes temporelles [Fargier97]. Il reste à fournir à l'auteur un moyen de percevoir globalement l'ensemble des scénarios possibles pour sa spécification du document (solutions au système de contraintes temporelles). Ceci fait l'objet d'un autre mémoire [Meurisse99].

2.6 Synthèse des approches d'édition temporelle

Résumons la situation. Muni de 15 critères, (selon l'expressivité du langage et les capacités d'édition), nous avons identifié deux familles : les approches opérationnelles et les approches par contrainte, et analysé chacune d'elle. Grâce à ces critères, nous allons comparer 4 logiciels d'édition temporelle multimédia : CMIFed, Isis, Firefly et Madeus. Le tableau comparatif détaillé se trouve en annexe A. Nous allons tirer de ce tableau quelques conclusions globales concernant les approches opérationnelles et par contraintes :

- Aucun outil ne fournit d'opérations fines sur les objets (causalité) : aucun ne distingue mapping et instant de début, ni unmapping et instant de fin. Pourtant, cette distinction

est nécessaire : un auteur peut vouloir afficher une vidéo à l'écran, et ne la démarrer que quelques secondes plus tard.

- Les effets de style et leur composition, pourtant très attrayants pour les auteurs, ont été négligés jusqu'à présent
- Il est difficile de comparer l'expressivité d'un langage, même avec la référence de la composition d'intervalles d'Allen. Par exemple, il faut pouvoir différencier une relation "Par_min" d'une relation de simultanéité.
- Les approches par contraintes sont mieux adaptées pour satisfaire les besoins des auteurs, car elles ont une approche plus relative qu'absolue (pas d'ordre dans la spécification, modifications locales aisées : critères 9 et 11).
- Les modèles de documents multimédia seront très utiles, mais ne sont guère considérés à l'heure actuelle
- Les approches par contraintes sont plus adaptées pour créer des outils d'édition puissants. En effet, ils offrent une plus grande expressivité. Le calcul des durées est pris en charge par le formateur, pas par l'auteur. Cela fournit des scénarios plus facilement réutilisables (traduction, etc). Notons que ces approches sont confrontées à de douloureux problèmes de performances durant la vérification de cohérence et le formatage. (Firefly a choisi la programmation linéaire, tandis que Isis et Madeus ont choisit les réseaux de contraintes temporelles). Les objets incontrôlables empirent la situation (Firefly et Madeus) lors de la phase de formatage.
- Les approches par contraintes sont mieux adaptées pour les présentations multimédia effectuées dans un environnement distribué. En effet, le réseau, sa charge et ses délais sont des problèmes inabordables pour les approches opérationnelles (désynchronisation, etc). Par contre, les systèmes de contraintes permettent de modifier dynamiquement une spécification, si cette dernière n'a pas été trop stricte.

En résumé, les approches par contraintes paraissent nettement plus intéressantes. Pourquoi donc? Et bien, tout provient de l'aspect relationnel. L'auteur pense en terme de relations entre objets, et ces approches utilisent explicitement les relations temporelles. Cela permet d'utiliser des positionnements relatifs, bien plus intéressants du point de vue de l'auteur d'un document. De plus, les contraintes numériques permettent une certaine souplesse dont les approches opérationnelle ne disposent pas. Pour clôturer le tout, les approches relationnelles par contrainte peuvent utiliser la dimension hiérarchique (logique) des documents, pour permettre la réutilisation de parties, etc. **En un mot, du point de vue d'un auteur de document, les approches relationnelles sont bien plus intéressantes.**

Par contre, elles posent de gros problèmes de performances, en terme de temps d'exécution. Ils seront abordés dans les chapitres ultérieurs et font actuellement l'objet de recherches. De manière assez amusante, nous pouvons constater que le même genre de conclusions ont été tirées lors de la comparaison des approches déclaratives en programmation (fonctionnelle et logique) avec les approches opérationnelles (impérative). Chaque approche a ses fervents admirateurs. Gageons qu'il en sera de même dans le domaine du multimédia.

Néanmoins, ce mémoire, assez objectif mais sans doute un peu biaisé, s'intéresse à une approche par contrainte : c'est le langage Madeus, qui est au cœur du logiciel Madison. Il fera l'objet du prochain chapitre. Mais juste avant, faisons un point sur les standards internationaux existant en matière de description de documents multimédia.

2.7 Les standards de l'édition

Avant d'étudier l'éditeur multimédia spécifique développé à l'INRIA, il faut considérer un dernier aspect de l'édition. Une fois un document créé, sa vocation est d'être distribué. Cela pose le problème de l'échange de tels documents. La situation ne peut être résolue efficacement que lorsque l'on dispose de standards. C'est l'objet ce cette dernière section.

La représentation des documents multimédia est un problème encore relativement ouvert. Il est nécessaire que s'imposent des standards pour commencer à y voir plus clair, et surtout pour rendre possible l'échange de documents multimédia. Hier assez rares, ces standards sont aujourd'hui assez nombreux. Il existe un service de la commission Européenne, l'OII (Open Information Interchange, [Standard-2]) chargé de fournir une vue générale des standards et spécifications facilitant l'échange de documents sous forme électronique. Nous y trouvons une liste [Standard-3] de 10 standards d'échange de documents multimédia et/ou hypermédia. Plus impressionnante encore (mais plus large dans sa sélection), une étude suisse [Standard-1] répertorie 40 standards associés d'assez près au multimédia.

Parmi tous ces standards, certains portent des noms pour le moins peu évocateurs : G.711, H.261, etc. Heureusement, nous retrouvons certaines connaissances : JPEG, MPEG, X400, MIME, GIF, QuickTime, MIDI. Nous en trouvons également certains plus directement en rapport avec le thème qui nous préoccupe :

- HyperODA (Hypermedia extensions to the Open Document Architecture)
- HyTime (Hypermedia/Time-based structuring language), ISO/IEC 10744:1997
- MHEG (Multimedia and Hypermedia coding Experts Group), ISO/IEC 13522
- SMIL (Synchronized Multimedia Integration Language)
- PREMO (Presentation Environment for Multimedia Objects)

Cette section examine brièvement certains de ces standards existants (HyTime, MHEG et SMIL), en les définissant et en situant leur utilisation actuelle. Quelques références bibliographiques pourront nous fournir de plus amples informations (section 2.7.4 page 37). Nous terminerons par une rapide conclusion à propos du niveau d'utilisation de tous ces standards.

2.7.1 HyTime

HyTime est le premier standard que nous abordons maintenant. Nous n'allons pas entrer dans les détails, par ailleurs bien trop complexes. Nous allons présenter son fonctionnement, puis dire un mot de son taux d'utilisation.

Définition

Dans le domaine des documents structurés, nous connaissons bien sûr le langage SGML (Standard Generalized Markup Language, ISO 8879). Son but est de représenter des documents structurés statiques. Pour cela, il utilise des marqueurs (*tags*)[7], et les DTD (Document Type Definition), qui définissent une classe de documents. Il restait à ajouter à cet édifice la dimension temporelle. SGML a donc été étendu aux documents hypermédia comprenant une dimension temporelle : HyTime, un nouveau standard international, était né. Il s'agit du "*Hypermédia Time-based structuring langage*". Il a été standardisé par l'ISO (International Organization for Standardisation) et l'IEC (International Electrotechnical Commission) en 1992.

7. d'où son nom de Markup Language ...

Une seconde édition a apporté des révisions techniques, en août 1997. HyTime est aujourd'hui connu sous le doux nom de "*ISO/IEC International Standard 10744:1997*".

HyTime est une application SGML. Il n'est pas une DTD en lui-même, mais fournit des constructions et des directives pour créer des DTD décrivant des documents hypermédia. HyTime permet évidemment de décrire la structure du document et les hyperliens entre éléments —il permet notamment d'utiliser les liens qui visent des cibles multiples. En outre, il permet d'exprimer des relations spatio-temporelles entre éléments d'un document.

Plus de détails

Les relations spatio-temporelles sont spécifiées par des coordonnées abstraites appelées FCS (Finite Coordinate Space) sur des axes. HyTime en prévoit autant qu'il en faut. Ainsi, les coordonnées d'un axe pourraient être le numéro d'ordre des éléments d'un document suivant un parcours en largeur d'abord de son arbre hiérarchique ! Cette capacité, quoique puissante, ne nous est guère utile : quatre axes (un spatial et trois temporels) nous suffiront amplement. Les valeurs réelles des coordonnées (secondes, etc) seront mises en correspondances avec ces FCS par une application compatible HyTime, capable de "jouer" un document HyTime. Il est permis d'effectuer un certain nombre de synchronisations grâce à ce langage. Ainsi, il existe une notion d'alignement (gauche, centre, droite). Leur signification est évidente dans la dimension spatiale. Dans la dimension temporelle, elle correspond à un début commun, à une mise en parallèle, et à une fin commune.

La spécification complète est assez compliquée et sort très largement du cadre de ce mémoire. D'ailleurs, la version imprimée comporte plus de 450 pages ! Pour faciliter la tâche, HyTime a été conçu de façon modulaire : il comporte 5 modules (en anglais : base, location address, hyperlinks, scheduling and rendition), dont seul le premier est obligatoire. Ce module (base) fournit des mécanismes pour les autres modules (description des limites d'un groupe d'objets). Le module "location address" utilise les FCS pour l'accès aux éléments. Le module "hyperlinks" permet la gestion des éléments de type lien. Le module "scheduling" planifie les éléments comme des événements d'une présentation (placement des objets selon leurs relations spatio-temporelles sur les FCS). Enfin, le module "rendition" possède le double rôle de modifier certains éléments (filtrage, etc) et de projeter les FCS sur des axes à unités "réelles". Heureusement, tous les modules ne doivent pas obligatoirement être utilisés pour obtenir un document conforme à la norme !

Utilisation

HyTime étant un standard complexe, il existe très peu d'implémentations qui l'utilisent. Une enquête [Standard-1] listait seulement deux projets de construction d'une application HyTime ! Un groupe d'utilisateurs HyTime [HyTime-3] recense quelques outils sur leur site [HyTime-4]. Mais il paraît que la situation évolue et que plusieurs projets vont utiliser des sous-ensembles de HyTime.

2.7.2 MHEG

MHEG est le second standard sur lequel nous nous penchons. Nous n'allons toujours pas rentrer dans les détails, trop complexes. Nous allons plutôt lister ses objectifs, le définir un peu plus, puis considérer son taux d'utilisation.

Ce standard est également l'œuvre de l'ISO et de l'IEC. Son autre petit nom est *ISO/IEC 13522*. MHEG est un acronyme pour *multimedia and Hypermedia information coding Expert Group*.

Objectifs

Le but recherché est multiple :

- Échange. D'abord, MHEG est un encodage conçu de manière à permettre l'échange de données multimédia à l'intérieur et entre applications.
- Présentation. Il s'agit du codage final de présentations hypermédia et multimédia, apte à interagir avec l'utilisateur pendant une présentation (taille, volume, etc). Des groupes d'éléments permettent une synchronisation dans le temps et dans l'espace.
- Ressources minimales. Les machines doivent le supporter avec un minimum de puissance CPU. Ceci est spécialement orienté vers les applications réseaux
- Temps réel. Les présentations et échanges d'informations multimédia doivent être adaptés au domaine du temps réel (visio-conférence)
- Enfin, une bonne portabilité et une compatibilité ascendante sont visés.

Définition

L'approche de MHEG est une approche orientée objet. Il utilise l'ASN.1 de l'ISO (Abstract Syntax Notation One). Nous y trouvons 4 types d'objet (classes). D'abord des *content objets*, représentant l'élément multimédia lui-même. Les encodages utilisés sont d'autres standards, comme JPEG ou MPEG. Des *Composite objets* permettent de créer des groupes d'objet. Ensuite, des *Link objets* sont des liens permettant de spécifier des contraintes spatio-temporelles ou conditionnelles entre objets MHEG. Enfin, les actions à effectuer sont spécifiées dans des *Action objets*. Ils spécifient les traitements à effectuer sur les objets, comme l'exécution ou l'arrêt.

Notons que MHEG utilise, comme HyTime, des coordonnées génériques pour les événements de synchronisation dans le temps et dans l'espace. Mais il se limite à 3 axes spatiaux (latitude, longitude, altitude pour traiter la 3D) et à un axe temporel, ce qui devrait être amplement suffisant. Un temps abstrait est également utilisé, avec comme unité des GTU (Generic Time Unit), en opposition à des PTU (Physical Time Unit).

MHEG permet de décrire un schéma de présentation entièrement basé sur les événements. Il est très orienté vers la description du flot de contrôle du document. Son principe de fonctionnement est le suivant : la présentation est gérée par l'échange de listes d'action objets permettant de décrire des présentations séquentielles ou parallèles. En utilisant les liens, nous obtenons une présentation entièrement basée sur des événements : des conditions d'activation quelconques peuvent en effet être attachées à ces événements.

MHEG est une norme qui a été définie et raffinée à plusieurs reprises. C'est un problème, car un MHEG devrait être plus stable pour assumer pleinement son statut de standard. Les parties intéressantes sont MHEG-5 (version de base, en client-serveur), MHEG-6 (extensions de programmation, avec API java pour MHEG-5, "iso.mheg5"), MHEG-7 (interopérabilité et tests, octobre 1998) et MHEG-8 (notation XML pour MHEG-5, mars 1999). Comme dans le cas de HyTime, tout n'est pas obligatoire : seuls certaines méthodes doivent être définies dans certaines classes. Par exemple, les méthodes "run" et "stop" des "action objects" sont obligatoires.

Utilisation

Les produits et services MHEG deviennent disponibles. Il existe entre autres un projet de recherche qui prépare un toolkit MHEG dans le cadre d'un projet européen. Quelques autres semblent se décider à l'adopter [Standard-3]. British Telecom a d'ailleurs développé une application de démonstration appelée MADE [Standard-1]. Depuis sa version 5, MHEG est bien adapté à la vidéo à la demande et aux services de téléachats. C'est dans ces environnements ouverts (réseaux) qu'il est le plus adapté. Cependant, la décision par le consortium MPEG d'adopter le "Media Control Interface" (MCI) de Microsoft a jeté un froid. En effet, MHEG a été spécialement créé pour les réseaux WAN (Wide Area Networks) : à quoi servira-t-il s'il utilise des standards propriétaires et valables sur une seule plateforme?

2.7.3 SMIL

SMIL est le dernier standard auquel nous nous intéressons. Il mérite notre attention, car Madison supporte ce langage aussi bien que Madeus. Ce n'est pas plus mal, car cela facilite les échanges de documents. Nous allons brièvement définir SMIL et son taux d'utilisation.

Définition

SMIL est un acronyme pour Synchronized Multimedia Integration Langage. Il a le statut de recommandation du W3C depuis juin 1998. SMIL a été pensé en considérant l'exemple de HTML : son succès est dû au fait que des documents hypertexte à contenu attrayant (images, fonds, etc) pouvaient être rédigés sans outils complexes — un simple éditeur de texte suffisait. SMIL a donc été créé en gardant cet objectif de simplicité. Ses spécifications existent en format BNF (Backus-Naur Form) et au format de DTD XML (Document Type Définition donnée en eXtended Markup Language).

SMIL possède les caractéristiques suivantes. Ses éléments sont accessibles via des URLs et sont de plusieurs types (audio, vidéo, image ou texte). Le lecteur d'un document SMIL dispose de boutons du style magnétoscope pour sa navigation temporelle ; il peut aussi choisir des instants spécifiques et y sauter immédiatement. Détail amusant, un document SMIL peut être joué à une vitesse plus lente que celle pour laquelle il a été créé. Les liens hypertextes sont évidemment activables. Terminons par le volet temporel. Les instants de début et fin de composants doivent être synchronisés avec des événements d'autres composants. Par exemple, un transparent d'une présentation s'affiche quand le commentaire audio qui l'accompagne commence à parler de ce transparent. Les opérateurs temporels en SMIL comportent la séquence (seq) et la simultanéité (par).

Utilisation

SMIL est employé par des développeurs d'outils audiovisuels, mais il reste relativement peu adopté par les créateurs de browsers web [Standard-3]. Malgré tout, plusieurs outils sont d'ores et déjà disponibles.

2.7.4 Références bibliographiques pour les standards

Nous venons d'examiner 3 standards. Il en existe d'autres, mais le but de ce mémoire n'est pas de les recencer tous. Les précédents ont été décrit sommairement. Pour obtenir de plus

amples informations, nous pouvons consulter les quelques références suivantes. Il s'agit d'une mini-bibliographie annotée.

SGML ISO 8879, XML, SMIL,...

SGML veut dire Standard Generalized Markup Language. C'est un standard soutenu par le W3C. Il a généré toute une famille de standards. Ainsi, XML (eXtended Markup Langage) en est un sous-ensemble, bien plus simple à implémenter et à utiliser. XML 1.0 a acquis le statut de "recommandation du W3C" en octobre 1998. Nous en saurons plus en parcourant les sites suivants :

- Le site du W3C, le Word Wide Web Consortium [Standard-4]
- Un site très bien fourni d'informations sur la famille SGML [Standard-5]

HyTime (ISO/IEC 10744:1997)

HyTime veut dire *Hypermedia Time-based structuring langage.* Plusieurs sites web, donnés en bibliographie, nous renseignent sur le standard Hytime :

- la norme [HyTime-1] proprement dite (document de 450 pages)
- ses liens avec le groupe SGML sur oasis-open [HyTime-2]
- un groupe d'utilisateurs HyTime [HyTime-3]
- une liste d'outils liés à HyTime [HyTime-4]

MHEG *(ISO/IEC 13522)*

MHEG veut dire *multimedia and Hypermedia information coding Expert Group.* Parmi les sites d'informations sur ce standard, nous pouvons consulter celui-ci :

- [Mheg-1], en particulier les sections "about, guided tour, faq, tools, bibliography, glossary, other sites"

SMIL

SMIL veut dire Synchronized Multimedia Intégration Langage. Quelques sites lui sont consacrés. Parmi eux, nous trouvons :

- la norme [Smil-1]
- un didacticiel SMIL écrit en SMIL [Smil-2]
- une applet java permettant de lire des documents SMIL [Smil-3]
- un outil d'édition de documents SMIL : "SMIL Composer" de Sausage Software [Smil-4]

2.7.5 Situation des standards

Après avoir défini sommairement les standards, il faut s'interroger sur leur taux d'utilisation, pour savoir où l'on se trouve en la matière. Force est de constater que la situation n'est guère brillante.

Les logiciels utilisant les standards sont encore très peu nombreux. Il n'est pas toujours facile de rédiger un outil conforme à ces normes, surtout si elles sont envisagées dans tous

leurs aspects et toute leur complexité. L'émergence d'un standard largement accepté semble encore assez utopique à l'heure actuelle. Les rédacteurs de standards supposent de plus que des outils performants existent pour transformer en structures exécutables leurs spécifications. Or, ce n'est pas forcément le cas. Du côté industriel, il existe beaucoup d'applications, mais leurs possibilités sont limitées par leur faibles études théoriques. Du côté de la recherche, c'est le contraire : il en existe assez, mais avec peu d'outils disponibles et finalisés.

Nous constatons qu'il reste beaucoup à faire dans le domaine des systèmes d'édition de documents multimédia. Madison en est une contribution, avec son langage Madeus. C'est ce logiciel, suivant une approche déclarative/par contraintes bien particulière, que nous allons étudier au prochain chapitre.

Chapitre 3

Madison et Madeus II

Nous nous sommes intéressés au chapitre précédent au techniques d'édition multimédia. Nous avons identifié les approches opérationnelles et les approches déclaratives. Ces dernières semblent bien plus intéressantes pour les auteurs. C'est pourquoi l'INRIA de Grenoble a opté pour une approche par contrainte. Elle a choisi de développer une application d'édition de documents multimédia structurés selon cette approche. Ce logiciel a été nommé Madeus, puis Madeus II, puis Madison[1].

Ce chapitre présente cette application, Madison. Nous définirons les lignes directrices qui orientent les choix effectués. Ensuite, nous définirons le langage Madeus, et examinerons la représentation d'un document et des relations. Nous spécifierons exactement quelles sont les relations utilisables dans Madeus. Enfin, nous terminerons par une note situant la complexité et la taille de Madison.

3.1 INRIA : Madeus et Madison

INRIA, Madeus, Madison : qu'est-ce, au juste? L'INRIA de Grenoble est un institut de recherche français dont un département a fait des documents structurés sa spécialité : le projet Opera. L'INRIA a d'ailleurs de nombreux contacts avec le Word Wide Web Consortium (W3C). Ainsi, un de ses membres, Nabyl Layaïda, fait partie du "synchronized multimedia working group" qui a préparé la rédaction des spécifications du langage SMIL [Smil-1]. Ces spécifications ont atteint le niveau de "recommandation du W3C" en juin 1998. Un éditeur de documents structurés (statiques) avait déjà vu le jour : Thot. Il restait à examiner la voie des documents multimédia structurés. Ce fut chose fait avec l'introduction de la dimension multimédia à l'INRIA par Nabyl, auteur d'une thèse fondatrice en la matière [Layaïda97]. Une application d'édition et de présentation de documents structurés allait voir le jour : Madeus.

Madeus 1.0 a été rédigé en langage C, après avoir effectué un certains nombres de choix présentés au long de ce chapitre. Un langage spécifique de spécification de documents multimédia structurés a spécialement été créé pour l'occasion. Ce langage, baptisé Madeus, a été défini selon une DTD (Document Type Definition) conforme au standard XML 1.0 du W3C, par raffinements successifs. Armé du langage Madeus et de l'application l'utilisant, des tests pratiques ont pu être effectués pour valider certaines hypothèses. Par la suite, pour des raisons de portabilité principalement, l'application a été réécrite entièrement en langage java (pour lequel un analyser/parser IBM était disponible), puis étendue : Madeus 2.0 voyait le jour.

1. Le nom définitif de la toolkit l'implémentant est maintenant déposé. Il s'agit de l'acronyme "KAOMI"

L'approche orientée objet de Madeus 2.0 lui confère un caractère suffisamment générique pour pouvoir être adapté en API ou toolkit. Ce travail est en cours de réalisation actuellement, en plus du développement proprement dit. Au passage, Madeus 2.0 a été rebaptisé Madison. Dans la suite de ce document, "Madeus" désignera le langage, tandis que Madison décrira l'application d'édition multimédia qui utilise ce langage comme base de spécification pour ses documents.

3.2 Orientations

Nous allons présenter tout d'abord les lignes directrices générales qui sont à l'origine de la création de Madison. Ces lignes directrices sont déterminées par les approches déclaratives, les capacités d'abstraction, la facilité pour l'utilisateur et des phases d'édition et présentation assez proches. Voyons cela.

Approche déclarative, par contraintes

Le projet Opéra de l'INRIA a adopté résolument pour une approche déclarative, par contraintes. Cette décision est amplement justifiée, au regard des avantages globaux de ce type d'approche (section 2.5 page 28). C'est la décision logique prise à la suite de la lecture de la synthèse des approches d'édition (sect. 2.6 page 32).

Capacité d'abstraction

Madison a été créé avec également en vue la capacité d'abstraction, de manière à garantir une réutilisation aisée d'éléments. Ceci constitue la dimension hiérarchique du document, décomposable sous forme d'arbre. Des méta-langages comme SGML ont réussi à promouvoir un modèle générique de tout document : ceux-ci peuvent être spécifiés selon trois dimensions indépendantes. (logique, spatiale, hypertexte). Ainsi, un même document peut être présenté spatialement de plusieurs manières différentes (c'est le principe des feuilles de style CSS). Madison ayant déjà adopté la première dimension, elle a fait de même avec les autres et en a ajouté une dernière, la dimension temporelle. Voici donc une nouvelle façon désormais classique de décomposer tout document selon quatre dimensions orthogonales :

1. Dimension logique : arbre hiérarchique du document, évoqué ci-dessus
2. Dimension spatiale : agencement des éléments sur le support (papier, écran)
3. Dimension hypertexte : liens bien connus, à la manière du Web
4. Dimension temporelle : agencement des éléments dans le temps

Dans la suite de cette section, nous allons examiner comment Madison spécifie ces dimensions dans son langage ad hoc : Madeus.

Facilité pour l'utilisateur

Un des buts primordiaux de Madison est la facilité d'édition multimédia pour un auteur. Presque tous les outils disponibles sur le marché utilisent une approche opérationnelle, ce qui les force à utiliser un langage de script pour combler les lacunes du paradigme sous-jacent. Or, la pratique de la programmation est loin d'être un acquis naturel de tout auteur profane en la matière. Nous ne pouvons même pas affirmer que cela soit évident pour les

informaticiens confirmés ; ce ne sont certainement pas les études alarmantes du génie logiciel en ce qui concerne la piètre qualité des "méthodologies" utilisées qui le démentiront. Bref, une des priorités principales de Madison est de ne surtout pas utiliser de langage de script pour arriver au résultat escompté. Cela serait d'ailleurs contraire à toute la philosophie des éditeurs structurés et des langages du style HTML.

Cet objectif est louable, bien sûr, mais il est extrêmement ambitieux. D'autre part, la simplicité pour l'utilisateur implique très souvent une complexité inouïe pour les développeurs. L'intrusion des interfaces graphiques dans le monde de la micro-informatique nous l'a démontré. Elle s'est introduite à un prix élevé. D'abord, cela a généré des besoins énormes en termes de puissance machine — essayez de faire tourner Windows 98 sur un AT-386 ... Ensuite, la simplicité offerte aux utilisateurs a compliqué la vie des informaticiens, car il a fallu tenir compte de bien des dimensions supplémentaires : ergonomie, interfaces homme-machine, application aux menus adaptables en fonction du niveau de l'utilisateur, systèmes d'aide hypertexte, modèles de documents, assistants, etc. La liste est longue et peut être étendue à l'infini.

Édition et présentation

Madison possède une caractéristique unique en son genre, à notre connaissance : elle tente de se rapprocher du Wysiwyg en intégrant le plus possible les phases d'édition et de présentation multimédia.

L'édition de documents statiques est possible en Wysiwyg. Par contre, elle ne l'est plus dès que nous introduisons une dimension temporelle : l'auteur ne peut spécifier un comportement dynamique et le voir s'exécuter en même temps. La première phase est la phase d'édition, de spécification du comportement multimédia souhaité pour un document. La seconde est la phase de présentation, où l'auteur visualise son document et interagit avec lui, pour vérifier que le document produit correspond à ses attentes. Ces deux phases sont intrinsèquement disjointes, ce qui interdit toute idée d'approche purement Wysiwyg.

Néanmoins, Madison tente l'impossible en rapprochant les deux le plus possible. Par exemple, l'auteur visualisant un scénario nouvellement créé peut s'apercevoir d'un détail gênant. Madison lui permet d'arrêter la présentation et de l'éditer directement. En fait, il s'agit d'un passage entre les deux modes, qui doit être le plus rapide possible — cela pose de terribles exigences de performances au niveau du temps de réponse du logiciel.

3.3 Langage Madeus : les quatre dimensions

Maintenant que nous connaissons les grandes lignes orientant le choix des décisions, voyons quel est le langage de description de document sous-jacent à Madison. Il suit la décomposition désormais classique en 4 dimensions : logique, spatiale, hypertexte et temporelle.

3.3.1 Dimension logique

La dimension logique d'un document est sa structure hiérarchique. Nous pouvons l'imaginer comme une structure d'arbre, à la manière des documents XML.

Eléments basiques

Les feuilles de cet arbre, ce sont les éléments de base d'un document Madeus ("basic objects"), càd les médias (texte, son, image, vidéo, applet, etc). Le langage Madeus utilise un système de marqueurs (tags) — il est d'ailleurs défini par une DTD conforme à XML. Chaque média est spécifié dans un document Madeus par un marqueur approprié. Ce dernier détermine le début et la fin d'un élément, à la manière des marqueurs HTML. Pour chaque élément, certains attributs sont définis. Ainsi, chaque média d'un document Madeus possède un attribut "Name" l'identifiant au sein de ce document. Un média possède aussi généralement un attribut "Source" dont la valeur est une URL fournissant l'adresse où trouver le média. Pour les médias "texte", cet attribut est omis au profit d'un attribut "Value" dont la valeur est le texte du média. Voici un exemple illustrant la spécification d'un média de type "son" : ce dernier est enregistré dans un fichier "brabançonne.au" du répertoire "/home/hawaii" local et sera identifié au sein du document par la chaîne "son1". Ceci se trouve en figure 3.1. Nous aurions pu l'écrire plus simplement, en utilisant un raccourci autorisé par XML qui respecte encore la DTD Madeus (figure 3.2).

```
<Audio
     Name="son1"
     Source="/home/hawaii/brabanconne.au"
     Duration="40 60 80" >
</Audio>
```

FIG. 3.1 – *Madeus : description d'un média (son)*

```
<Audio Name="son1" (...) Duration="40 60 80" />
```

FIG. 3.2 – *Madeus : description d'un média (son), version abrégée usuelle*

Nous pouvons remarquer que d'autres attributs sont disponibles, selon le type de média. Dans l'exemple précédent, un attribut "Duration" supplémentaire était mentionné. Il voulait simplement dire que le son devait être joué entre 40 et 80 unités de temps, avec une préférence pour une durée de 60 unités.

Ces attributs additionnels d'un élément basique (un média) servent à spécifier entre autre des attributs spatiaux. Par exemple, les attributs "Left" et "Top" fixent la position du coin supérieur gauche d'un média image. D'autres attributs sont des attributs temporels. Par exemple, l'attribut "Duration" est commun à tous les médias. Il permet de spécifier la durée désirée pour le média, selon le format traditionnel du triplet $[m, \lambda, M]$, désignant respectivement les valeurs minimale (m), préférentielle (λ) ou maximale (M) que pourra prendre la durée de cet objet lorsque le document sera présenté. Avant d'en terminer avec la description des éléments basiques, rappelons-nous que la sémantique exacte du triplet $[m, \lambda\, M]$ dépend du type de média utilisé (cfr. section 2.5.2 page 29).

Elements composites

Voilà pour les éléments basiques, les médias. Les éléments composites, eux, servent à donner la structure hiérarchique du document, en groupant (en une séquence non vide) des éléments basiques ou composites. En voici la syntaxe. Un élément Madeus composite est un élément doté de quelques attributs (Name, Duration, Top, Left, etc). Son contenu est une suite non

vide d'éléments basiques ou composites, suivis d'un élément "Relations". Voici un exemple d'un composite groupant un son, deux images et un autre composite. C'est la figure 3.3.

```
<Composite Name="Intro" Duration="40 60 80" >

    <Audio Name="son1" (...) Duration="40 60 80" />
    <Picture Name="img1" (...) Duration="20 30 40" />
    <Composite (...)> (...) </Composite>
    <Picture Name="img2" (...) Duration="20 30 40" />

    <Relations> (...) </Relations>

</Composite>
```

FIG. 3.3 – *Madeus : description d'un composite (sauf ses relations)*

Voilà comment l'auteur spécifie la dimension logique de son document. Bien sûr, les exemples donnés ici sont incomplets et ne donnent pas tous les détails. En particulier, la spécification des relations a été passée sous silence. Elles seront développées dans les dimensions spatiales et temporelles, en reprenant ce même exemple. La véritable définition des éléments et attributs utilisables, ainsi que leur sémantique précise sortent du cadre de ce mémoire. Pour plus d'information, la source d'information officielle est bien sûr la DTD Madeus en vigueur à l'INRIA. Pour la comprendre, il nous suffit de connaître la syntaxe et les principes de XML, qui ne sont pas très compliqués. La DTD utilisée par Madison évolue régulièrement, mais l'une de ses anciennes versions figure en annexe B page 113.

3.3.2 Dimension spatiale

La dimension spatiale d'un document Madeus est spécifiée par composant (basique ou composite), afin de rester indépendante de la dimension logique. Elle est donnée en deux parties. La première façon de donner des informations de nature spatiale est de fixer des attributs de position absolue aux médias en tant que simples attributs (comme Top et Left dans les exemples précédents). La seconde manière concerne les positions relatives entre éléments.

Dans la dimension temporelle, Madison a opté pour une spécification de ces relations sous forme déclarative plutôt que procédurale. Pour le même genre de raisons, Madison procède de même dans la dimension spatiale. Les relations utilisables sont simples et doivent être vérifiées tout au long de la présentation du document. Nous pouvons spécifier des relations du style "A à gauche de B", "A et B centrés horizontalement", "A situé à 50 pixels à gauche de B", etc. Ces relations existent toutes en version horizontale et verticale ; elles sont aussi bien de nature symbolique (gauche, droite, etc) que numérique (15 pixels, etc).

En reprenant notre précédent document exemple de la page 45, nous voulons spécifier en Madeus que l'image 1 se situe à gauche de l'image 2, avec la relation "Left_align"[2]. Notre petit document exemple peut être spécifié comme montré sur la figure 3.4.

Le premier problème des relations entre objet consiste à déterminer à quel endroit elles peuvent être définies. Si nous permettons l'insertion de relations entre n'importe quel couple de nœuds dans le graphe hiérarchique, cela pose des problèmes relativement complexes (détection d'incohérences, dépendances cycliques dans les relations, etc). Pour contourner ces difficultés

2. La liste complète des relations utilisables est consultable via la DTD de Madeus (annexe B à la page 113).

```
<Composite Name="Intro" Duration="40 60 80" >

    <Audio Name="son1" (...) Duration="40 60 80" />
    <Picture Name="img1" (...) Duration="20 30 40" />
    <Composite (...)> (...) </Composite>
    <Picture Name="img2" (...) Duration="20 30 40" />

    <Relations>
        <Spatial>
            <Left_align Interval1="img1" Interval2="img2" />
        </Spatial>
    </Relations>

</Composite>
```

FIG. 3.4 – *Madeus : description d'un composite avec relations spatiales*

sans perdre trop du pouvoir d'expression des relations temporelles, Madison a opté pour un système légèrement plus restrictif : les relations spatiales ne peuvent être ajoutées qu'entre deux éléments faisant partie du même composite. Autrement dit, elles n'existent qu'entre deux nœuds de même père. Ainsi, la gestion de ces relations est simplifiée tout en conservant un haut niveau d'expressivité.

Le second inconvénient associé à la spécification de relations spatiales par le biais de contraintes est la phase de formatage et la vérification de cohérence associés. En effet, il faut pouvoir détecter les éventuelles incohérences dans la spécification. Par exemple, les relations "A à gauche de B" et "B à gauche de A" sont incompatibles : elles ne peuvent être vérifiées en même temps si nous considérons ces relations sous leur forme stricte[3]. La phase de formatage, elle, consiste à trouver (au moins) une solution possible satisfaisant l'ensemble des contraintes. Ici, il s'agit de définir une position pour chaque élément (y compris les composites) de telle manière que tous les souhaits de l'auteur soient respectés. Concrètement, le logiciel d'édition doit inclure un résolveur de contraintes. Celui employé dans Madison est Deltablue, un logiciel développé aux États-Unis en langage C — il suffisait de l'intégrer à l'architecture java.

Le dernier inconvénient à contourner est celui du problème des performances, évoqué à la section 3.2 page 42. Pour obtenir des temps d'exécution très courts, il faut disposer d'algorithmes au point et très optimisés. Au pire, il faut se contenter de caractéristiques moins puissantes, en réduisant par exemple l'expressivité du langage des contraintes spatiales, de manière à diminuer la complexité que l'algorithme doit gérer.

Néanmoins, tous ses problèmes restent gérables, et le résultat est encourageant. Une utilisation bien réussie de ce système dans Madison concerne le passage rapide entre édition et présentation. Voyons-le sur un exemple. Supposons qu'un objet A soit soumis à une contrainte le forçant à rester à la même abscisse (coordonnée horizontale). Supposons également qu'un objet B soit soumis à une contrainte le forçant à avoir la même ordonnée (coordonnée verticale) que A. Madison maintient les contraintes spatiales en permanence, dynamiquement, à la présentation du document. Ceci fournit une très bonne illustration des contraintes pour

3. Sous leur forme "large", ce n'est pas le cas : A et B seraient simplement en même position horizontale, (càd centrés horizontalement sur un axe vertical, ou superposés). Ce genre de raisonnement à base de relations larges est souvent utilisé dans les démonstrations mathématiques pour prouver l'égalité ou l'équivalence de deux êtres mathématiques. ($a \leq b$ et $b \leq a \Rightarrow a = b$)

une personne sceptique. Pendant la présentation, cette personne peut cliquer sur l'objet A. Ce faisant, la première contrainte intervient: A ne peut bouger que dans un couloir vertical. La personne pourra également cliquer sur l'objet B et le faire bouger de haut en bas, mettant en jeu la seconde contrainte: l'objet A suit le mouvement vertical de B. Mieux encore, en cliquant sur l'objet B et en le faisant bouger dans toutes les directions, cette personne met en jeu les deux contraintes simultanément: il verra que l'objet A suit l'objet B verticalement, mais en restant dans son couloir vertical. Cette expérience, menée pendant une présentation, démontre la puissance des systèmes à programmation par contraintes, bien supérieure aux approches opérationnelles. En dehors de l'exploit technique de maintenir le respect des contraintes dynamiquement, ceci peut convaincre quelqu'un de manière simple et conviviale.

Du côté des relations spatiales, l'emploi du résolveur Deltablue se montre satisfaisant, y compris dans ses temps de réactions, comme l'a illustré l'exemple précédent. Nous ne reviendrons plus sur cet aspect, même s'il est responsable en grande partie de la beauté "plastique" d'un document multimédia: notre préoccupation sera concentrée sur les aspects temporels.

3.3.3 Dimension hypertexte

La dimension hypertexte concerne les systèmes de liens, devenus classiques via HTML sur Internet. Un lien peut être un simple renvoi à un autre élément, à la manière d'un renvoi à une bibliographie dans un document imprimé. HTML a étendu légèrement le champ d'application aux liens "à travers" un livre: les liens vers d'autres parties de document, activables par simple clic. Ceci confère aux documents une espèce de dimension sémantique (réduite, bien sûr). Cela semble bien simple et connu de tous. Mais tout n'a pas été dit à ce sujet.

Le système de lien précédent ne renvoie qu'à un élément. Parfois, il est souhaitable de pouvoir désigner par un lien plusieurs éléments. Ces "liens multiples" sont bien plus riches: un lecteur intéressé par un lien pourra choisir parmi ceux désignés celui qui est le plus proche de ce qu'il désire. Un bon exemple serait une entrée d'index dans un système d'aide. Une entrée d'index est un terme auquel le lecteurs s'intéresse, dans le but de trouver rapidement une information qui lui est associée. Or, une entrée d'index renvoie souvent à de nombreux sujets. Par exemple, une entrée "note de bas de page" d'un système d'aide d'un traitement de texte doit pouvoir renvoyer à plusieurs informations: sa création, sa suppression, sa mise en page, etc Donc, les liens multiples sont très utiles. La notion de lien est actuellement soumise à des standardisations qui envisagent cet aspect parmi d'autres.

Nous pouvons aller encore bien plus loin. Les liens ont été considérés jusqu'ici dans leur dimension spatiale. Mais rien n'empêche de les utiliser dans leur dimension temporelle. Un lien doit pouvoir signifier "aller directement 3 secondes plus loin dans la présentation", "recommencer la présentation de la section 5", etc.

Ces aspects méritent une attention toute particulière, au bénéfice de l'auteur et du lecteur d'un document multimédia. Le lecteur d'un document multimédia préfère les documents hautement interactifs aux autres. Prenons l'exemple d'une personne consultant régulièrement un CD-ROM contenant une petite encyclopédie du corps humain. Supposons que chaque utilisation commence par une longue séquence présentant le fournisseur du logiciel. Cette personne ne supportera pas longtemps d'être réduit à supporter cette séquence sous prétexte qu'elle n'est pas interruptible par un bouton d'interaction. Il est probable qu'il s'en détournera pour un produit plus souple et plus interactif.

Ces aspects méritent encore notre attention dans le domaine de la navigation. Notre société n'est plus confrontée à une pénurie d'informations, mais plutôt à leur prolifération, qui rend

l'accès à une information bien précise plus difficile. Pour trouver la perle rare enfouie sous des masses d'informations peu porteuses du sens recherché, il faut disposer d'un système de recherche et de navigation efficace. C'est lors de la navigation que les liens hypertexte interviennent lourdement. Qui ne s'est pas retrouvé un jour perdu sur le web, après avoir navigué d'un site à l'autre, en ayant oublié le trajet suivi, voire même l'endroit d'où il était parti? Ceci est lié en partie à la capacité limitée de notre mémoire à court terme, mais cela ne met pas pour autant le système informatique hors de cause. Le concepteur d'un document doit veiller à ce que ces liens soient suffisamment expressifs, assez nombreux tout en demeurant structurés. Le fait de disposer d'un bon système de liens hypertexte peut de toute évidence être un atout décisif dans la comparaison de deux éditeurs de documents multimédia, y compris dans la comparaison des documents produits — ceci devrait intéresser nos amis les commerciaux. Notons quand même que des liens intelligemment répartis et spécifiés au sein d'un document n'est pas suffisant en soi. Le système de navigation présentant le document doit pouvoir aider l'auteur par des historiques de navigation ou autres systèmes ingénieux qui tardent à venir.

En un mot, nous constatons que la dimension hypertexte est trop souvent négligée, car elle est cruciale pour l'acceptation ou non d'un document par un lecteur. Quoique très intéressant à explorer, surtout de nos jours, ces aspects ne seront pas approfondis plus longuement dans le présent mémoire — ce n'est pas son objectif principal.

3.3.4 Dimension temporelle

La dimention temporelle est la dernière et la moins explorée d'un document. Beaucoup de choses ont déjà été évoquées dans la dimension spatiale et restent valables ici. La spécification de relations temporelles respecte également la structure hiérarchique du document. Ces relations ne peuvent être données que pour des éléments d'un même composite, un même père. Ceci simplifie la gestion de ces contraintes.

La dimension temporelle est spécifie en langage Madeus en deux parties. La première est l'attribut "Duration" associé à chaque média, sous la forme du traditionnel triplet $[m, \lambda, M]$ donnant l'intervalle de durées acceptable pour la présentation du média. Il faut cependant remarquer que la signification précise du triplet $[m, \lambda, M]$ est légèrement différente selon le type de média (cfr. section 2.5.2 page 29). La seconde partie concerne les relations temporelles relatives entre éléments. Ces relations sont du type symbolique ("A en même temps que B", etc) ou numérique ("A précède B de 12 secondes", etc).

En reprenant l'exemple du document-exemple précédent (page 45), spécifions que l'image 2 doit apparaître 10 secondes avant l'image 1, et que le son accompagne uniquement l'image 1. Nous pouvons l'exprimer en Madeus comme l'illustre la figure 3.5. Ceci n'est qu'un exemple : seule la DTD décrit entièrement les relations temporelles spécifiables en Madeus (annexe B à la page 113).

Les relations précédentes étaient une généralisation aux aspects numériques des relations d'intervalles d'Allen [Allen91]. Ceci sera spécifié en plus grands détails dans la suite de cet ouvrage. Il existe un autre type de relations : celui des relations de causalité ou d'interruption. Par exemple, la relation "A Par_min B", qui signifie que le plus court des deux termine l'autre, même si ce dernier n'était pas arrivé au bout de sa présentation. Ces relations posent problème, car il faudra différencier les instants de fin prévues et effectives, pour chaque élément.

L'approche par contrainte impose de vérifier la cohérence du système de contraintes temporelles, et de trouver (au moins) une solution à ce système, pour pouvoir le présenter. Le même souci de performances est présent. Le problème des intervalles incontrôlables, abordé

```
<Composite Name="Intro" Duration="40 60 80" >

    <Audio Name="son1" (...) Duration="40 60 80" />
    <Picture Name="img1" (...) Duration="20 30 40" />
    <Composite (...)> (...) </Composite>
    <Picture Name="img2" (...) Duration="20 30 40" />

    <Relations>
        <Spatial>
            <Left_align Interval1="img1" Interval2="img2" />
        </Spatial>
        <Temporal>
            <Before Interval1="img2" Interval2="img1" Delay="10" />
            <Equals Interval1="son1" Interval2="img1" />
        </Temporal>
    </Relations>

</Composite>
```

FIG. 3.5 – *Madeus : description d'un composite avec relations temporelles*

en fin de ce mémoire, complique encore la situation. Nous aborderons bientôt tout ceci plus en détail.

3.4 Représentation d'un document

Nous venons de présenter le logiciel Madison, quelques-unes de ses orientations générales, et son langage sous-jacent, Madeus. Mais il faut détailler beaucoup plus pour bien comprendre Madison. Cette section examine comment Madison gère la représentation d'un document, de manière suffisamment abstraite pour ne pas rentrer dans les détails d'implémentation (hiérarchie de classes, etc). Nous nous limiterons cependant assez bien à la partie qui concerne la dimension temporelle, en éclipsant quelque peu les autres. Comment Madison représente-t-il en interne un document et sa dimension temporelle? C'est ce que nous allons voir.

3.4.1 Informations mémorisées

Madison enregistre le document selon plusieurs dimensions. La hiérarchie du document (dimension logique) est mémorisée dans une structure d'arbre classique, aménagée pour l'occasion. A cet arbre sont associées énormément d'informations. Les relations entre éléments d'un composite sont mémorisées au niveau de ce composite. Pour chaque média, Madison retient une information temporelle de durée, sous forme d'un triplet $[m, \lambda, M]$ dont la sémantique a été donnée en section 2.5.2 page 29. Madison mémorise également les instants de début et de fin de chaque élément, basique ou non.

3.4.2 Graphe d'instants ou de contraintes

Tout ceci nous amène à la structure de données que Madison utilise pour représenter un document. Celle qui a été retenue est un graphe d'instants, qui a été étendu pour obtenir un graphe hiérarchique d'instants. Nous l'appellerons également graphe de contraintes, plus

loin dans le texte. En fait, un graphe d'instants est associé à chaque élément composite d'un document. Si un composite en comporte un autre, alors un arc du graphe du composite père contiendra une référence au graphe représentant le composite fils. Bref, chaque composite (chaque niveau hiérarchique) d'un document est représenté par un graphe d'instants.

Les sommets de ce graphe représente un instant temporel fixe. Chaque média est introduit dans le graphe de manière simple. D'abord, deux sommets correspondant aux instants de début et de fin du média sont crées. Ensuite, un arc reliant ces deux points est inséré dans le graphe. Cet arc représente le média ; il possède une étiquette composée d'un identifiant et d'un triplet de durée. La figure 3.6 illustre un média A ainsi représenté.

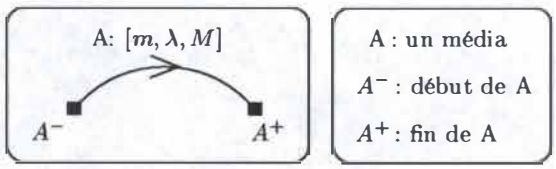


FIG. 3.6 – *Madeus : représentation d'un média A dans le graphe temporel*

Il faut ensuite représenter les relations entre objets. Ceci sera détaillé plus loin, car tout n'est pas si simple. Mais en un mot, il nous suffit de savoir que l'insertion d'une relation dans le graphe correspond à combiner entre eux les sommets et les arcs des médias concernés par la relation, en ajoutant éventuellement des arcs de délai, pour tenir compte des relations incluant des informations numériques. Par exemple, la figure 3.7 montre deux médias (A et B) devant être présentés l'un à la suite de l'autre, sans délai. La partie droite de cette figure montre un média A précédant un autre (B) de quelques instants (délai).

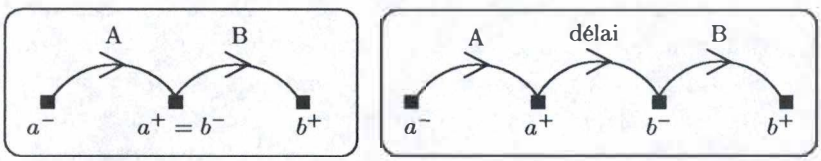


FIG. 3.7 – *Madeus : représentation de deux médias A et B en séquence puis avec relation de précédence*

Il reste à ajouter l'aspect hiérarchique à notre description. Pour représenter un composite dans un graphe, Madison le considère comme un média ordinaire en créant un arc. Il étiquette cet arc de manière à se rappeler qu'il s'agit d'un composite. De plus, il associe à cet arc une référence vers un autre graphe temporel : celui qui représente le composite.

3.4.3 Pour aller plus loin

La création du graphe et ses modifications, notamment lors de l'insertion d'éléments ou de relations, sera décrite plus loin. Cela se situe dans le chapitre consacré à l'implémentation de PC2, à la section 7.2 page 86.

3.5 Algèbres des relations temporelles

Nous avons vu comment représenter les éléments (médias) d'un document. Il nous faut voir comment spécifier des relations entre ces éléments.

Nous allons maintenant détailler ce qui se cache derrière les relations temporelles utilisées, avant d'examiner plus exactement la notion de contrainte. Nous disposons de deux grandes manières d'utiliser des relations temporelles entre objets : les relations à base d'instants et les relations à base d'intervalles. Nous les définissons dans cette section. Ensuite, nous expliquons celles qui ont été retenues pour Madison.

3.5.1 Relations à base d'instants

Une première manière de spécifier des relations temporelles est l'utilisation de relations à base d'instants. Ici, les objets sont considérés selon les sommets qui les représentent dans le graphe. Les relations temporelles sont uniquement celles définies entre les instants particuliers des éléments : leurs instants de début et de fin. En nous plaçant sur une ligne du temps, nous pouvons comparer deux instants selon leur position relative. Ce sont les trois relations primitives d'instants :

1. $<$: la précédence temporelle stricte (avant)
2. $=$: la simultanéité temporelle (en même temps que)
3. $>$: la postériorité temporelle stricte (après)

Si nous désirons spécifier des relations plus complexes, nous pouvons combiner par un "ou inclusif" ces 3 relations. Nous obtenons donc 2^3 relations composées, en passant à l'extension aux parties de cet ensemble :

1. $\perp$: ensemble "vide" de relations : il désigne une incohérence
2. $<$, $=$ et $>$: ensemble "singleton" : ce sont les 3 relations primitives
3. $\leq$, $\neq$ et $\geq$: ensemble "paire de relations" : précédence, différence et succession (inégalités larges)
4. ? : ensemble comprenant toutes les 3 relations primitives : aucune contrainte temporelle

En ce qui concerne l'édition multimédia, ces 8 relations ne sont pas toutes utiles. Ainsi, les relations "$\leq$", "$\neq$" et "$\geq$" diffèrent des relations primitives "$<$", "$=$" et "$>$" par un seul instant. Si l'unité de temps est la milliseconde, une telle différence ne sera pas perceptible. De plus, elle risque de se révéler impossible à garantir par le système de présentation. Donc, les seules relations d'instants utiles dans un système d'édition multimédia sont "$<$", "$=$", "$>$" et "?".

Lorsque nous parlons de relations à base d'instants, nous adoptons le vocabulaire d'usage : "algèbre d'instants" ou "point algebra" (PA). Cette appellation se justifie par des propriétés structurelles de l'ensemble des relations composées, muni des opérations d'union, d'intersection et de composition. Ce dernier opérateur est la loi de composition interne $\otimes$ définie selon le tableau suivant (son neutre est "$=$") :

3.5.2 Relations à base d'intervalles

Une autre manière de spécifier des relations temporelles entre éléments consiste à utiliser des relations à base d'intervalles. Ici, les objets sont considérés selon les arcs qui les représentent

$\otimes$	$<$	$=$	$>$
$<$	$<$	$<$	?
$=$	$<$	$=$	$>$
$>$	?	$>$	$>$

TAB. 3.1 – *Loi de composition $\otimes$ pour les relations d'instants*

dans le graphe. Les éléments à comparer sont donc des intervalles. Les différentes relations possibles entre deux intervalles ont été étudiées dans [Allen91] : les relations primitives sont les 13 relations d'Allen. Il s'agit de 7 relations de base et de leurs inverses. Le quatorzième élément manquant est l'égalité, qui est son propre inverse. Le tableau de la figure 3.8 (page 52) les définit de manière précise. Il est possible de traduire chaque relation d'intervalles en plusieurs relations d'instants. Ceci est également repris sur la figure. La notation x^- désigne l'instant de début d'un intervalle x, tandis que la notation x^+ désigne son instant de fin. Enfin, notons que les extensions numériques (paramètre t sur la figure) ne font initialement pas partie des relations définies par Allen.

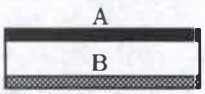
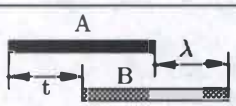
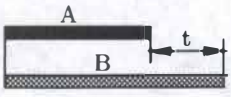
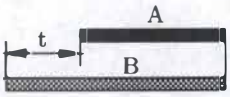
Sémantique graphique	Relations	Traduction en relations d'instant	Relations inverses
	A equals B	$a^- = b^- < a^+ = b^+$	B equals A
	A before(t) B	$a^- < a^+ < b^- < b^+$	B after(t) A
	A during(t) B	$b^- < a^- < a^+ < b^+$	B contains(t) A
	A overlaps(t) B	$a^- < b^- < a^+ < b^+$	B overlaped_by(t) A
	A meets B	$a^- < a^+ < b^- < b^+$	B met_by(t) A
	A starts(t) B	$a^- = b^- < a^+ < b^+$	B started_by(t) A
	A finishes(t) B	$b^- < a^- < b^+ = a^+$	B finished_by(t) A

FIG. 3.8 – *Les relations à base d'intervalles (Allen)*

En vue de spécifier des relations plus complexes, nous pouvons combiner par un "ou inclusif" ces 13 relations, en passant à l'ensemble des parties. Nous obtenons donc 2^{13}, soit 8192 relations composées: le pouvoir d'expression est bien plus élevé. Une des relations supplémentaires obtenues est celle qui demande que 2 objets A et B ne soient pas présentés simultanément. Autrement dit, A précède B (avec ou sans délai), ou bien B précède A (avec ou sans délai). Si nous désignons les primitives d'intervalles par leur initiale et leur inverse en leur adjoignant la lettre "i", ceci se note $(A\,b\,B) \vee (A\,m\,B) \vee (A\,bi\,B) \vee (A\,mi\,B)$. De manière plus concise, cette contrainte s'écrit $A\{b, m, bi, mi\}B$.

Pour des raisons similaires à celles utilisées pour les relations à base d'instants, les relations (composées) à base d'intervalles, munies des opérations usuelles, sont réunies sous le vocable d'algèbre d'intervalles (Interval Algebra: IA).

3.5.3 Relations mixtes

Une dernière manière de spécifier des relations entre éléments consiste à mélanger les deux manières précédentes (PA et IA). Nous allons les comparer, puis justifier le choix effectué dans Madeus.

Comparaison de PA et IA

Il est légitime de se baser sur les deux classes précédentes. Pour mesurer leur pouvoir d'expression, nous comparons les relations disjonctives de PA et de IA. En se limitant à une partie des relations composées de PA (instants), nous mesurons le nombre de relations composées de IA (intervalles) qui sont spécifiables. Consultons le tableau 3.2.

Relations à base d'instants de PA	Nombre de disjonctions de IA générables
<,=, >	13
<,=, >, ?	29
<,=, >, ?, ≤, ≥	82
<,=, >, ?, ≤, ≥, ≠	187

TAB. 3.2 – *Relations disjonctives de IA générées par celles de PA*

En consultant ce tableau, nous constatons que les relations disjonctives (composées) à base d'intervalles (IA) sont bien plus riches que celles à base d'instants (PA). Nous appellerons *RIA* (Restricted Interval Algebra) les 187 relations générables à partir de IA. La RIA n'est pas capable d'exprimer $8192 - 187 = 8004$ relations. Parmi celles-ci, nous retrouvons la relation d'exclusion de parallélisme citée plus haut, $A\{b, m, bi, mi\}B$, signifiant que les objets A et B ne peuvent être présentés simultanément. Ceci pourrait être utile dans le domaine du multimédia, si l'on sait par exemple que ces deux objets nécessitent une même ressource matérielle (haut-parleur, etc). Néanmoins, certains auteurs considèrent que les seules 29 relations (ligne deux du tableau) sont vraiment utiles au multimédia.

Choix de Madison

Les intervalles conviennent mieux au niveau descriptif: les objets sont représentés par des intervalles, et les niveaux hiérarchiques par des intervalles spécialement marqués. En outre, un auteur moyen préfère penser en terme de placement d'intervalles sur une ligne du temps.

Par contre, les instants sont plus adaptés au niveau de la présentation, car ils permettent l'utilisation d'une structure de graphe. C'est pourquoi Madison a choisi une approche mixte. Au niveau du langage, les relations spécifiables sont celles des intervalles. Au niveau de l'implémentation, une structure de graphe est utilisée. Celle-ci est construite en transformant en relations d'instants chaque relation d'intervalles, en vue de relier correctement les objets dans le graphe.

3.6 Types de relations temporelles

Nous utilisons des relations d'intervalles et des relations d'instants dans Madeus. Mais il existe plusieurs types de relations temporelles : qualitatives, quantitatives et causales. Les voici.

3.6.1 Relations qualitatives

Madison utilise la RIA, càd les 187 relations disjonctives d'intervalles générables par des relations disjonctives d'instants. Grâce à celles-ci, l'auteur peut placer les éléments dans le temps, sans se soucier de contraintes numériques. Ce sont par exemple les relations d'Allen telles qu'il les a définies, càd sans paramètre numérique.

3.6.2 Relations quantitatives

Ce sont celles qui autorisent des paramètres numériques. Madison a choisi d'étendre les relations d'Allen en adjoignant un paramètre temporel à certaines de ces relations. Ce type de relation est primordial pour l'auteur : il doit pouvoir ordonner qu'un objet A doit être présenté 10 secondes avant un objet B. Ce sont ces deux premiers types de relations qui étaient reprises dans la figure 3.8 page 52.

3.6.3 Relations causales

Ce sont les relations autrement connues sous le nom de relations d'interruption. Il peut arriver qu'un instant de fin d'un objet provoque la fin d'un autre. Ainsi, la fin d'une vidéo peut forcer la fin d'un autre objet, comme le titre (texte) qui décrit la vidéo. Ce type de relations résout un problème évoqué plus haut. Certains outils ont un problème avec leur opérateur de simultanéité. Supposons que A et B soient des vidéos à présenter en même temps, donc liées par une relation "A equals B". Si la durée de A est plus courte que celle de B, faut-il couper B ou laisser afficher A en attendant que B soit terminé? Ce problème provient de la non-distinction entre l'égalité et une relation causale.

Les relations causales que Madison utilise sont les suivantes :

- **A Par_min B** : A et B démarrent en même temps, et le plus court termine la construction. L'un des deux objets peut être tronqué.
- **A Par_max B** : A et B démarrent en même temps, et le plus long termine le tout. Aucun des deux objets ne sera tronqué.
- **A Par_master B** : A et B démarrent en même temps. A termine la construction. B sera tronqué si sa durée est supérieure à celle de A, sinon il sera présenté intégralement, en restant éventuellement présent à l'écran tant que A n'est pas terminé.

Ces relations disposent d'une interprétation graphique, en fonction de la durée relative des médias A et B. Le fait de ne pas savoir à l'avance lequel des deux médias sera le plus court va poser des problèmes délicats lors de la vérification de cohérence et du formatage. En effet, ces valeurs ne sont pas nécessairement connues avant la présentation du document. Il faudra donc un formatage dynamique, et un mécanisme de vérification de cohérence adapté. Nous en parlerons dans le chapitre 5 (page 67) consacré à la vérification de la cohérence. Dans ce chapitre, nous trouvons l'interprétation graphique de ces relations (page 69).

3.7 Le logiciel Madeus

Nous en avons terminé avec les relations spécifiables dans Madeus. Ces relations définissent des contraintes dont il faudra valider la cohérence. Ensuite, il faudra résoudre le système de contraintes pour présenter le document à l'écran. Les chapitres suivants présentent la gestion des systèmes de contraintes, ainsi que ce que l'on entend plus exactement sous le terme "cohérence". Mais avant cela, disons quelques mots sur le logiciel Madison.

Le logiciel a d'abord été rédigé en langage C, puis en Java. La taille du logiciel est conséquente : il compte environ 70 000 lignes de code. Mais notons bien que ce nombre élevé inclut la documentation complète du code source au format HTML, selon la philosophie de Javadoc. La partie gérant le graphe comporte environ 8 000 lignes. Quant aux résolveurs de contraintes, ils totalisent 2 500 lignes, alors que le résolveur de contraintes spatiales, développé en langage C, n'est pas repris dans le total ! Ceci donne une idée de l'ampleur du logiciel.

Pour mener à bien un tel projet, il est nécessaire de disposer de bonnes méthodes de développement. L'aspect orienté objet de java ainsi que ses très nombreuses librairies fournies constituent une aide précieuse. Lors de mon stage, j'ai pu constater l'efficacité redoutable de CVS, un logiciel de gestion de versions : il permet à plusieurs développeurs de travailler sur le même ensemble de fichiers, en ne se souciant quasiment pas de l'intégration du fruit de leur labeur.

Nous avons décrit Madison et ses principes, sans en voir d'images. C'est ce par quoi nous allons terminer ce chapitre. La figure 3.9 est une capture d'écran effectuée lors de l'édition d'un document multimédia hiérarchique avec Madison. Ce logiciel utilise le concept des vues multiples : le document est représenté simultanément sous divers angles synchronisés. Une modification effectuée dans l'une des vues est aussitôt répercutée dans les autres. La présentation du document à l'écran est effectuée dans la vue d'exécution. La hiérarchie du document est représentée sous forme d'arbre dans la vue hiérarchique. Une vue textuelle liste le code source du document (en langage Madeus). Enfin, une vue scénario montre le document à l'auteur sous une forme proche d'une ligne du temps. Dans cette vue, l'auteur peut modifier la durée d'un élément et voir dans quelle mesure il peut le faire en gardant une solution aux contraintes, car un intervalle rouge apparaît et y limite ses déplacements. Ceci fait l'objet d'un autre mémoire [Meurisse99].

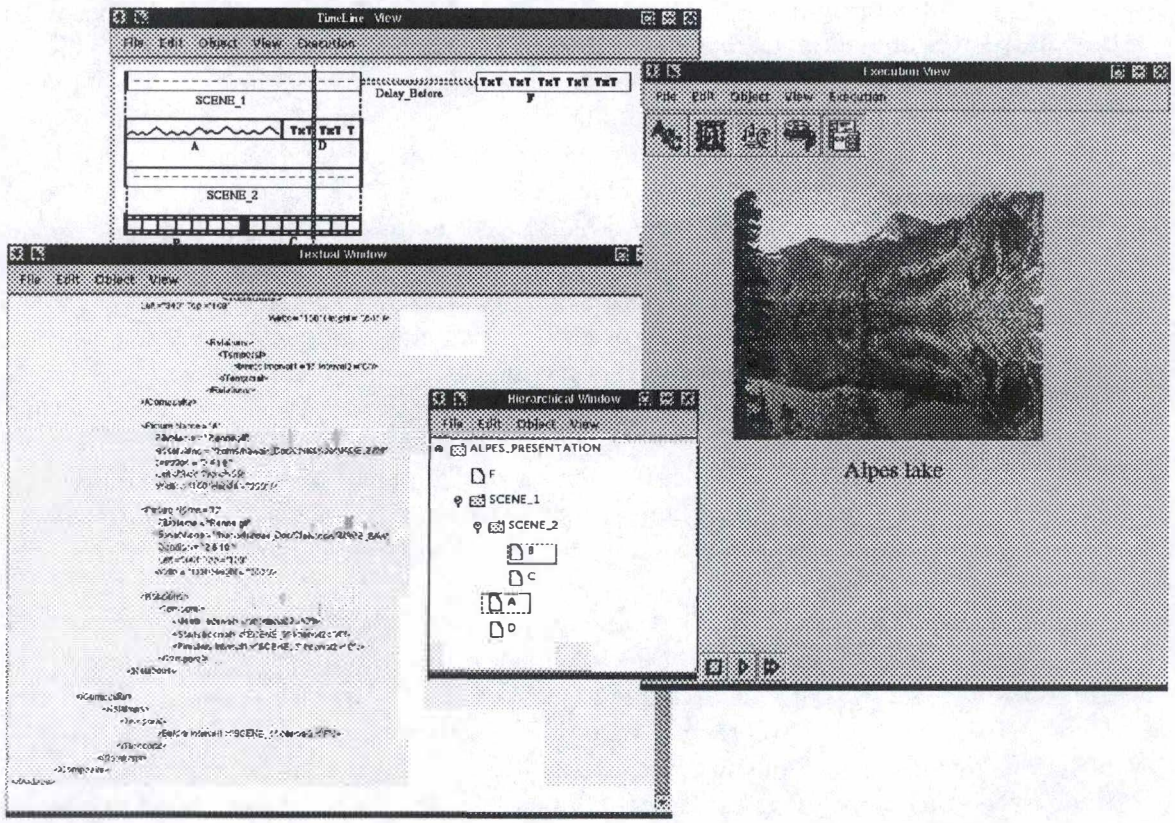


FIG. 3.9 – *Madison. Capture d'écran lors de l'édition d'un document multimédia hiérarchique. Il présente une vue scénario (en haut à gauche), une vue exécution (à droite), une vue hiérarchique (au milieu) et une vue textuelle (en bas à gauche)*

Chapitre 4

Contraintes temporelles

Nous avons opté via les chapitres précédent pour une approche à base de contraintes, plus intéressante pour les auteurs. L'INRIA a choisi de développer un logiciel, Madison, qui suit cette approche. Nous allons donc parler plus de contraintes temporelles, dans ce chapitre.

Il existe de nombreux types de systèmes de contraintes, ainsi que de nombreuses méthodes de résolution. Nous allons définir les types de systèmes de contraintes et préciser auquel nous avons à faire, à savoir les STP[1]. Il s'agit du cas simple, qui est déjà implémenté à l'heure actuelle.

4.1 Définitions

Nous parlons d'une approche par contraintes, et nous n'avons toujours pas défini ce que nous entendions par là. Ceci sera chose faite sous peu.

La démarche adoptée par Madison utilise un cas particulier des problèmes de résolution de contraintes, les CSP (Constraint Satisfaction Problems). Un outil utilisant les CSP doit fournir un moyen d'exprimer les contraintes ainsi qu'un moyen de les résoudre.

4.1.1 CSP

Un *CSP* est un problème de satisfaction de contraintes. Plus précisément, il est donné par un ensemble de variables sur lesquelles sont imposées des contraintes. Une solution de ce CSP sera une instanciation de toutes les variables qui respecte toutes les contraintes à la fois. Nous n'avons pas besoin des CSP dans toute leur généralité : les CSP binaires nous suffiront. Ce sont les CSP dont les contraintes sont spécifiées seulement entre deux variables. De manière plus précise et plus formelle, un CSP binaire est un quadruplet (X, D, C, R) défini par :

- $X = \{X_1, \ldots X_n\}$, un ensemble de variables
- $D = \{D_1, \ldots D_n\}$, un ensemble de domaines (D_i est le domaine de la variable X_i)
- $C = \{C_{i,j}\}$ où $1 \leq i, j \leq n$: un ensemble de contraintes liant chaque couple de variables
- $R = \{R_{i,j}\}$ où $1 \leq i, j \leq n$: un ensemble de relations – la relation $R_{i,j}$ est l'ensemble des couples de valeurs des variables X_i, X_j qui satisfont la contrainte $C_{i,j}$.

1. Dans la suite du développement de Madison, il nous faudra sans doute dépasser ce cadre trop restreint. En effet, la présence d'une hiérarchie d'objets, de relations de causalité et d'intervalles incontrôlables posent des problèmes dont certains restent ouverts à ce jour.

Une solution d'un CSP, σ, est définie par :

- $\sigma = \{x_1, \ldots x_n\}$ tels que $x_i \in D_i$ et $(x_i, x_j) \in R_{i,j}$ $\forall C_{i,j} \in C$, $\forall i, j \in \{1, \ldots n\}$

Si une telle solution existe, le CSP est dit *cohérent.* Un autre élément est intéressant à connaître. Si les domaines associés à un CSP sont restreints aux seules valeurs appartenant à au moins une solution, le CSP est dit *minimal.*

4.1.2 TCSP

Une sous-classe des CSP retient notre attention : ceux comportant une dimension temporelle, les *TCSP* (Temporal Constraint Satisfaction Problem). Leurs variables modélisent des instants ou intervalles temporels et leurs contraintes modélisent le placement temporel relatif de ces variables.

Comme spécifié plus haut, deux types de contraintes temporelles existent : les relations qualitatives (symboliques) ou quantitatives (numériques). Il existe des techniques bien adaptées à chaque cas. La vérification de la cohérence et le calcul d'une solution devront pouvoir prendre en compte les deux aspects.

4.1.3 TCSP numériques

Attardons-nous sur les contraintes numériques. Les contraintes temporelles peuvent s'exprimer comme des différences entre dates d'instants. C'est cette façon de faire que nous allons retenir, car les contraintes symboliques pourront s'exprimer de la même manière. Voici donc un TCSP binaire particulier : le *TCSP numérique.* Ceux-ci sont définis par un quadruplet (X, D, C, R) ayant quelques particularités :

- $X = \{X_1, \ldots X_n\}$. Chaque variable X_i représente un instant, une date
- $D = \{D_1, \ldots D_n\}$. Chaque domaine D_i est en fait $\mathcal{N}$, car les instants X_i sont des entiers positifs.
- $C = \{C_{i,j}\}$ où $1 \leq i, j \leq n$: Les contraintes (binaires) sont exprimées sous forme de disjonction de différences bornées, du type :

$$a^1_{i,j} \leq X_j - X_i \leq b^1_{i,j} \vee \ldots \vee a^n_{i,j} \leq X_j - X_i \leq b^n_{i,j}$$

- $R = \{R_{i,j}\}$ où $1 \leq i, j \leq n$: Les relations définissant les couples X_i, X_j de variables satisfaisant la contrainte $C_{i,j}$ sont ici un ensemble $I = \{I^1_{i,j}, \ldots, I^n_{i,j}\}$ d'intervalles de validité $I^k_{i,j} = [a^k_{i,j}, b^k_{i,j}]$.

Ceci signifie qu'un élément multimédia est modélisé par une union d'intervalles de validité. Ceci est plus général que ce que Madison a choisi, càd la modélisation d'un élément multimédia par un seul intervalle de validité. Ceci est motivé par le fait que les algorithmes qui résolvent les TCSP à base d'union d'intervalles ne sont pas polynômiaux. Nous choisissons donc de restreindre ces unions d'intervalles de validité à un seul élément : c'est ce que l'on appelle les *STP* (Simple Temporal Problem).

4.1.4 STP

Un STP est donc bien plus simple, comme son nom l'indique d'ailleurs. Ses algorithmes de résolution se mesurent en temps polynômial. Dans un contexte d'édition interactive où les exigences en performances sont primordiales, c'est le strict minimum de disposer d'algorithmes polynômiaux. Il faudrait même qu'ils soient plus performants (cubiques, carrés ou linéaires par exemple). Un STP est défini par un quadruplet (X, D, C, R) on ne peut plus simple :

- $X = \{X_1, \ldots X_n\}$. Chaque variable X_i représente un instant, une date
- $D = \{D_1, \ldots D_n\}$. Chaque domaine D_i est en fait $\mathcal{N}$, car les instants X_i sont des entiers positifs.
- $C = \{C_{i,j}\}$ où $1 \leq i, j \leq n$: Les contraintes (binaires) sont exprimées sous forme d'une différence bornée, du type : $a_{i,j} \leq X_j - X_i \leq b_{i,j}$
- $R = \{R_{i,j}\}$ où $1 \leq i, j \leq n$: Chaque relation $R_{i,j}$ définissant les couples X_i, X_j de variables satisfaisant la contrainte $C_{i,j}$ est un intervalle de validité $I_{i,j} = [a_{i,j}, b_{i,j}]$.

C'est dans ce cadre a priori très restreint que nous allons nous cantonner dorénavant. Chaque média sera modélisé par un intervalle de validité $I_{i,j} = [a_{i,j}, b_{i,j}]$ et chaque contrainte numérique sera de la forme $C_{i,j} = a_{i,j} \leq X_j - X_i \leq b_{i,j}$.

Étant donné une contrainte $C_{i,j}$, nous pouvons aisément déduire sa contrainte associée, $C_{j,i}$. En effet, $C_{i,j}$ peut s'exprimer comme une paire d'inégalités $a_{i,j} \leq X_j - X_i$ et $X_j - X_i \leq b_{i,j}$. En la réécrivant autrement, nous obtenons $-b_{i,j} \leq X_i - X_j$ et $X_i - X_j \leq -a_{i,j}$. La contrainte $C_{j,i}$ est donc déduite : $C_{j,i} = -b_{i,j} \leq X_i - X_j \leq -a_{i,j}$. Cette espèce de symétrie va nous aider à définir un graphe de distances bornées.

4.1.5 STP et graphes

Les STP ont la propriété intéressante de pouvoir être exprimés sous forme de graphes. Voyons cela dans cette section.

Représentons chaque variable X_i (chaque instant) par un sommet i. Par simplicité, ajoutons deux sommets spéciaux X_0 et X_{n+1}, symbolisant respectivement le début et la fin du scénario multimédia. Ensuite, représentons chaque intervalle de validité $I_{i,j} = [a_{i,j}, b_{i,j}]$ par un arc entre les sommets i et j, avec $[a_{i,j}, b_{i,j}]$ comme étiquette. Il faudra ajouter les intervalles de validité $I_{0,j} = [a_{O,j}, b_{0,j}]$ et $I_{i,n+1} = [a_{i,n+1}, b_{i,n+1}]$ pour déterminer quels seront les premiers et derniers médias à être présentés, par rapport au début et à la fin du scénario. Le graphe obtenu représente totalement notre système de contraintes. C'est d'ailleurs ce graphe qui est utilisé dans Madison pour la résolution des contraintes temporelles. La symétrie évoquée précédemment nous autorise à définir un graphe légèrement différent de celui que nous venons de décrire. Il est appelé graphe de distance temporelle.

Un *graphe de distance temporelle* $G_d = (X, A, E)$ est défini par :

- les sommets $0, \ldots n$ représentant les variables $\{X_0, \ldots X_n\}$
- les arcs $A = \{A_{i,j}\}$ où chaque arc $A_{i,j} : X_i \rightarrow X_j$ relie deux sommets distincts i et j.
- les étiquettes $E = \{E_{i,j}\}$. Soit $[a_{i,j}, b_{i,j}]$ l'intervalle de validité associé à la distance entre X_i et X_j. Alors, l'arc $A_{i,j} : X_i \rightarrow X_j$ possède l'étiquette $E_{i,j} = b_{i,j}$ et l'arc $A_{j,i} : X_j \rightarrow X_i$ reçoit l'étiquette $E_{j,i} = -a_{i,j}$.

Remarquons la signification des étiquettes dans ce graphe :

- L'arc orienté joignant i à j, $X_i \xrightarrow{b_{i,j}} X_j$, signifie que $X_j - X_i \leq b_{i,j}$.

- L'arc orienté joignant j à i, $X_i \xleftarrow{-a_{i,j}} X_j$ signifie que $X_j - X_i \geq a_{i,j}$, ou $X_i - X_j \leq -a_{i,j}$.
- La sémantique reste donc la même, quelle que soit la direction des arcs.

4.1.6 Autres définitions liées aux STP

Certains autres concepts peuvent être intéressants dans le domaine des STP:

- Deux problèmes STP sont dits *équivalents* s'ils représentent le même ensemble de solutions.
- Un STP est dit *minimal* s'il n'existe aucun STP équivalent dont les contraintes soient plus restrictives. Nous verrons un algorithme fournissant les domaines minimaux, donc un STP minimal équivalent au STP initial. Il s'appelle PC2.
- Un STP est dit *décomposable* si chaque instantiation d'une valeur de durée pour un élément n'est jamais remise en cause lors du choix des valeurs des durées des autres éléments du scénario.

4.2 Théorèmes concernant les STP

Pour résoudre les STP, quelques résultats théoriques sont les bienvenus. En voici..

Il s'agit de trouver les valeurs à attribuer aux instants $X_1, \ldots X_n$ de manière à respecter toutes les contraintes. En notant X_0 l'instant de début d'un scénario et en lui attribuant la valeur 0, la résolution du système revient à trouver les valeurs entières des distances X_j à X_0 $(= 0)$. Voici quelques théorèmes pouvant nous informer pour cette tâche.

4.2.1 CNS pour la cohérence d'un système

Nous commencerons par un théorème fournissant une Condition Nécessaire et Suffisante (CNS) pour établir la cohérence d'un système, via son graphe de distance. Il s'appuie sur la remarque suivante. Soit un chemin de sommets $i_0, \ldots i_k$ reliant les sommets $i = i_0$ à $j = i_k$. En notant $c_{p,q}$ le poids (étiquette) associée à l'arc joignant p à q, nous obtenons une contrainte temporelle entre i et j :

$$X_j - X_i \leq \sum_{t=1}^{k} c_{i_{t-1},i_t}$$

S'il existe plusieurs chemins de i à j, nous notons $d_{i,j}$ la durée du plus court chemin entre i et j. Ceci nous amène à l'observation qui est à la base du théorème suivant :

$$X_j - X_i \leq d_{i,j}$$

Théorème 1 *Un STP est cohérent si et seulement si son graphe de distance G_d ne contient pas de cycles négatifs.*

La preuve est construite à partir de l'observation précédente ; elle n'est pas bien compliquée. La voici.

Preuve 1 *Démonstration du théorème précédent*

⇒ *Par absurde, supposons qu'il existe un cycle négatif (càd dont la somme des étiquettes est strictement négative). Soient* $i_1, i_2, \ldots, i_k = i_1$ *les sommets formant ce cycle. La somme des inégalités* $X_j - X_i \leq d_{i,j}$ *le long de ce circuit nous donne* $X_{i1} - X_{i1} < 0$. *Ceci n'est pas possible.*

⇐ *Comme il n'existe pas de cycle négatif, le plus court chemin entre chaque paire de sommets est bien défini. Soit* $d_{0,i}$ *la distance de* 0 *à* i *sur le chemin le plus court les joignant. Notons* $c_{i,j}$ *l'étiquette joignant le sommet* i *au sommet* j. *Alors, l'égalité* $d_{o,j} \leq d_{0,i} + c_{i,j}$ *est satisfaite. D'où* $d_{0,j} - d_{0,i} \leq c_{i,j}$. *Le couple* $(X_j = d_{0,j}, X_i = d_{0,i})$ *vérifie donc la contrainte binaire entre* i *et* j. *Nous avons donc trouvé une solution au STP :* $(d_{0,1}, \ldots, d_{0,n})$.

4.2.2 Construction de solutions particulières

Il existe un corollaire dérivé du théorème précédent qui donne deux solutions généralement distinctes à un STP.

En utilisant un raisonnement similaire à celui de la preuve précédente à partir de l'instant de fin plutôt que de l'instant de début du scénario, on obtient $d_{j,0} - d_{i,0} \leq c_{j,i}$, ce qui se réécrit en $(-d_{i,0}) - (-d_{j,0}) \leq a_{j,i}$: le couple $(X_i = -d_{i,0}, X_j = -d_{j,0})$ vérifie donc la contrainte entre i et j. Ceci donne lieu à un corollaire fournissant deux solutions pour les valeurs des variables.

Corrolaire 1 *Soit* G_d *le graphe de distance d'un STP cohérent. Alors, il existe au moins deux scénarios cohérents. Ce sont* $S_1 = (d_{0,1}, \ldots, d_{0,n})$ *et* $S_2 = (-d_{1,0}, \ldots, -d_{n,0})$. S_1 *est le scénario aux dates d'occurence au plus tard* $(c_{i,j} = b_{i,j})$, *tandis que* S_2 *est le scénario aux dates d'occurence au plus tôt* $(c_{i,j} = -a_{i,j})$.

4.2.3 Décomposabilité

Enfin, voici un théorème de la plus haute importance pour la suite. Il justifie la correction de l'algorithme employé dans Madison lors de la présentation d'un document. En effet, la présentation d'un document nécessite le calcul d'une solution au système des contraintes temporelles. Si le STP est décomposable, cela signifie que nous pouvons choisir une valeur de durée correcte pour un intervalle (dans le domaine minimal), sans devoir remettre en cause ce choix plus tard. Le système de présentation de Madison se base sur le fait que tout STP est décomposable, lorsque l'on considère les domaines minimaux. Et le théorème suivant nous assure que c'est bien le cas.

Théorème 2 *Tout problème STP est décomposable par rapport aux contraintes décrites par son graphe minimal de distance*

La preuve de ce théorème n'est pas tellement compliquée. Le lecteur intéressé la trouvera dans [Layaïda97, p. 71].

4.2.4 Intérêt de la décomposabilité des STP

D'un point de vue pragmatique, la propriété de décomposabilité des STP est utile pour Madison à divers égards :

- Le *formatage* du document est simplifié et accéléré, car aucun retour en arrière n'est requis lors du calcul d'une solution aux contraintes temporelles

- Il est possible de définir plusieurs *politiques de priorité* lors du choix d'une solution, car l'ordre d'instanciation des variables n'a pas d'importance. Ainsi, nous pourrions commencer par fixer les durées des éléments sensibles comme l'audio ou la vidéo, de manière à s'approcher le plus possible des valeurs préférentielles de l'auteur.
- Cela facilitera sans doute le formatage lorsqu'il deviendra dynamique, en vue par exemple de tenir compte des *intervalles incontrôlables.*

4.3 Résoudre un STP

Nous utilisons des STP dans Madison. Nous savons ce qu'ils sont et connaissons des théorèmes et propriétés les affectant. Il faut maintenant aborder le problème de leur résolution : si la solution n'est pas calculable en des temps raisonnables, l'approche par contraintes serait vaine.

Résoudre le STP revient à déterminer les valeurs des variables $X_0, \ldots, X_n$ (qui sont des entiers symbolisant des instants). Ces variables doivent être positives ($X_i \in \mathcal{N}\ \forall i \in \{1, \ldots, n\}$), car le temps ne s'écoule pas à reculons. De plus, elles doivent vérifient simultanément toutes les contraintes $a_{i,j} \leq X_j - X_i \leq b_{i,j}$. Il s'agit donc de résoudre un ensemble d'inéquations linéraires sur des variables entières positives. Plusieurs classes d'algorithmes peuvent nous y aider. En voici un aperçu.

4.3.1 Résolution via la programmation linéaire

Une première façon de résoudre un STP utilise la recherche opérationnelle, au croisement des mathématiques et de l'informatique. Elle connaît bien les systèmes linéaires d'inégalités. L'algorithme classique du simplex permet de les résoudre en un temps exponentiel. Il optimise une fonction linéaire sur un domaine borné exprimé au moyen de ce type de contraintes (égalités ou inégalités, peu importe). Pour l'utiliser, il suffit de définir une fonction à optimiser, car il n'y en a pas a priori. Pour la phase de formatage, une solution simple consiste à trouver la solution s'écartant le moins possible des valeurs préférentielles définies par l'auteur pour les durées. Si la solution n'est pas trouvée, c'est que le système de contraintes est incohérent. L'algorithme du simplex présente des inconvénients gênants dans le domaine de l'analyse et de la génération de scénario.

Le premier concerne les domaines minimaux. En effet, cette méthode nous présente une solution globale lorsqu'elle existe, mais ne nous fournit pas de manière explicite les domaines minimaux. L'auteur ne sait pas dans quelle mesure il peut modifier les valeurs obtenues pour que les variables modifiées restent une solution au problème.

Le second concerne l'identification des incohérences. L'algorithme de simplex rejette tout en bloc en cas d'incohérence, sans fournir d'indication sur les contraintes en cause. Mais ceci est un problème survenant dès que l'on utilise des contraintes. Il s'agit d'un problème quasi insoluble. En effet, si un système devient cohérent après l'insertion d'une contrainte, il est fréquent que ce soit une conjonction de plusieurs contraintes qui soient incompatibles. Comment savoir lesquelles sont en cause, et où arrêter la recherche des "responsables" ? Ceci semble être un problème ouvert et assez complexe.

Un dernier inconvénient concerne le temps d'exécution. Cet algorithme est général : il ne tient pas compte de la forme très spécifique des contraintes d'un STP. Son temps d'exécution est exponentiel, même s'il semble se comporter de manière plus efficace dans la pratique.

Beaucoup plus grave encore, il n'est pas incrémental : il faut tout recommencer dès l'insertion d'une nouvelle contrainte ou la modification de certaines paramètres.

Cette approche a été choisie par l'équipe de développement d'Isis (section 2.5.5 page 30) et celle de Firefly (section 2.5.6 page 31). Mais ce style d'approche n'est guère adapté à un processus d'édition incrémental. D'ailleurs, à cause du temps d'exécution trop pénalisant du simplex, l'équipe d'Isis s'est orientée vers une approche basée sur les réseaux de contraintes.

4.3.2 Résolution via les réseaux de contraintes (graphes)

Une autre façon de résoudre les STP utilie des graphes. Ces approches tiennent compte de la forme particulière des contraintes temporelles. Elles l'exploitent pour structurer les contraintes sous forme de graphe. C'est ainsi que nous avons défini le graphe de distance temporelle G_d (section 4.1.5 page 59). C'est possible dans le cadre restreint des STP, comme nous l'avons vu. Nous verrons ici deux genres d'algorithmes. Le premier, celui du plus court chemin, utilise le graphe sans considération particulière. Le second manipule directement les intervalles de validité (PC2, DPC-incrémental).

L'algorithmique sera appliquée à une structure de graphe acyclique orienté. La marche inexorable du temps explique le caractère orienté[2] et acyclique[3] du graphe.

Le fait de tenir compte des spécificités des contraintes et d'utiliser une algorithmique de graphe confère des qualités recherchées dans l'édition multimédia. En effet, les temps d'exécution sont sensiblement meilleurs. De plus, il existe des algorithmes ad hoc qui sont incrémentaux. Les exigences de performance en matière d'édition temporelle multimédia incrémentale sont telles que cette approche risque de supplanter l'autre. Isis a déjà fait le pas, d'ailleurs.

Algorithme utilisant les graphes

Parmi les algorithmes utilisant les graphes, il y a ceux le faisant à l'aveugle, sans considérer de manière spéciale les étiquettes de durée y attachées. Nous avons vu dans la section 4.2 page 60 un théorème nous fournissant deux solutions (ordonnancement au plus tôt et au plus tard). Ces solutions étaient définies par les valeurs de durées de chemins les plus courts, sur un graphe sans cycle. C'est pourquoi un algorithme de recherche de chemins le plus court suffit ici. L'algorithme du plus court chemin de Floyd-Warshal (all pairs shortest paths) convient parfaitement. Il est polynômial pour déterminer la cohérence d'un STP. Ensuite, l'instanciation d'une solution au cours du formatage est en $O(n^2)$, car le choix d'un instantiation doit être conforme aux précédentes, sans être modifiées ultérieurement. Voici donc l'algorithme en question.

Un exemple Comme un dessin vaut bien mieux que de longs discours, voici un exemple illustré. Il provient de [Layaïda97, page 68]. Les graphes représentant le problème se trouvent sur la figure 4.2 page 64. Lorsque nous exécutons l'algorithme de Warshal sur le graphe de distance, nous obtenons le tableau 4.1. Nous en concluons que les dates au plus tôt et au plus tard sont les suivantes : $X = \{10, 40, 20, 60\}$ et $X = \{20, 50, 30, 70\}$. Ces deux solutions sont représentées sur la figure 4.3 page 65.

2. Les voyages dans le passé ne sont pas encore au point de nos jours ...
3. Notre futur n'est pas encore passé, semble-t-il ;-)

Pour i de 1 à n faire $d_{i,i} \leftarrow 0$
Pour i et j de 1 à n faire $d_{i,j} \leftarrow a_{i,j}$
Pour k de 1 à n faire
    Pour i, j de 1 à n faire
        $d_{i,j} \leftarrow \min\{d_{i,j}, d_{i,k} + d_{k,j}\}$

FIG. 4.1 – *Résolution des STP par graphes : algorithme du plus court chemin (Warshal)*

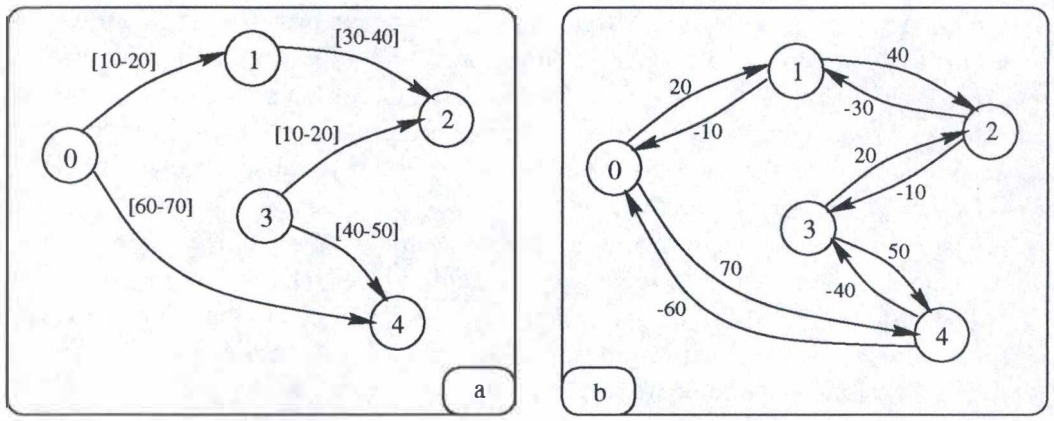


FIG. 4.2 – *Algorithme de Warshal : graphe de contraintes (a) et graphe de distance (b)*

Algorithme manipulant les intervalles de validité

Une deuxième classe d'algorithme résolvant les STP à l'aide de graphe manipule directement les intervalles de validité. Nous pouvons citer un algorithme mis au point par Allen, PC2 et DPC incrémental, qui sera vu dans le chapitre consacré à la cohérence, à la section 5.3.3 page 72.

Les algorithmes de cette classe utilisent un graphe comme structure de données. L'algorithme utilisé dans Madison est appelé PC2 (Path Consistency, deuxième du nom). Celui-ci n'utilise plus un graphe temporel tel que nous l'avons défini. Nous avons appelé ce nouveau type de graphe "graphe de contraintes". Ses arcs sont étiquetés par les intervalles de validité, et plus par des entiers (positifs dans un sens et négatifs dans l'autre).

Allen avait proposé un autre algorithme, assez proche : il étiquetait les arcs par les relations

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

TAB. 4.1 – *Algorithme de Warshal : résultats pour le graphe de distance*

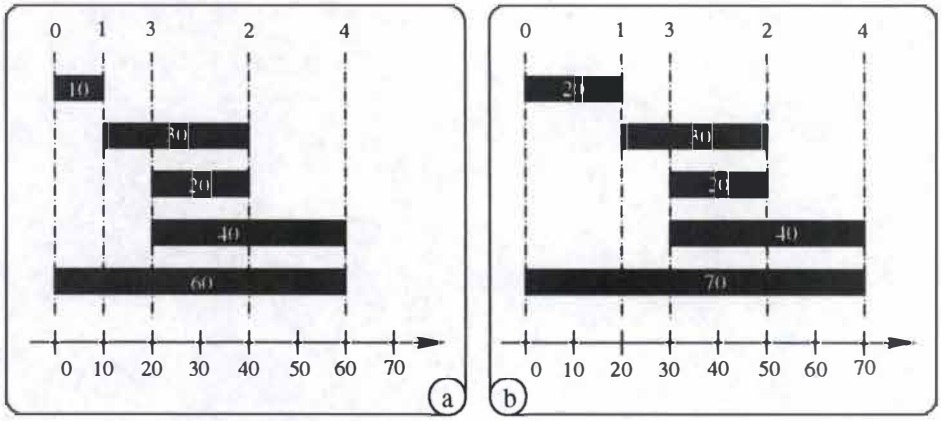


FIG. 4.3 – *Algorithme de Warshal : résultats graphiques sous forme d'ordonnancement au plus tôt (a) et au plus tard (b)*

temporelles, puis utilisait une technique de propagation. Mais il l'appliquait aux relations d'intervalles, plus expressives que les relations d'instants. Le prix à payer en est l'incomplétude (certaines incohérences ne sont pas détectables) et la complexité (le temps d'exécution est trop long).

Madison a une fois de plus une approche mixte. PC2 utilise une technique de propagation sur les intervalles étiquetant les arcs. Les relations que le graphe temporel comporte proviennent de RIA, les 187 relations d'intervalles générables via les disjonctions de relations d'instants. Cela fournit un pouvoir d'expression moindre, mais largement suffisant. De plus, le temps d'exécution de cet algorithme s'en voit diminuer. En dernier lieu, cet algorithme ad hoc tient compte de la hiérarchie des documents.

Un chapitre ultérieur traitera en détail de PC2. Il s'agit de l'algorithme que j'ai implémenté durant mon stage à l'INRIA. Mais avant cela, examinons d'un peu plus près ce que nous entendons par le terme "cohérence".

Chapitre 5

Vérification de cohérence et formatage

Nous avons opté pour Madison, un logiciel d'édition de documents multimédias structurés suivant une approche par contraintes. Nous avons examiné ce qu'étaient ces contraintes, ainsi que certains principes aidant à leur résolution. Il nous faudra résoudre ces systèmes de contraintes. Mais avant de les résoudre, il faut savoir s'ils ont ou non une solution. S'ils n'en ont pas, c'est que le système de départ était impossible, incohérent. Il faudra pouvoir déterminer si un système est cohérent ou non. Et il faudra savoir ce que l'on entend exactement sous le terme "cohérence". C'est un concept à multiples facettes que nous examinons dans ce chapitre.

L'auteur d'un document multimédia structuré est confronté à des problèmes d'incohérence, dès qu'il utilise un éditeur à base de contraintes. Ceci est vrai dans la dimension spatiale comme dans la dimension temporelle. C'est à cette dernière que nous nous intéressons ici. Nous commencerons par définir plus précisément ce que nous entendons par cohérence en donnant des définitions générales. Puis, en section 68 page 5.2, nous en désignerons les différents types : quantitative, qualitative, causale et indéterministe. Toutes ses définitions acquises, nous allons enfin pouvoir parler précisément de la vérification de cohérence, en section 5.3 page 71. Nous donnerons les exigences à respecter, le nom de deux vérificateurs de cohérence et une liste d'extensions à apporter par la suite. Nous avons parlé de l'édition de documents. Nous dirons un mot de leur présentation, càd du formatage. Il consiste à trouver une ou plusieurs solutions au système de contraintes, en vue de présenter un document au lecteur. Les mécanismes de vérification de cohérence peuvent être réutilisés pour formater un document. Ceci constitue la section 5.4 page 73, qui clôture ce chapitre.

5.1 Cohérence : définitions

Commençons par définir plus précisément ce que nous entendons par cohérence. Il en existe en fait de plusieurs types. La théorie nous apporte quelques définitions. Nous pouvons distinguer les cohérences locales (deux premières définitions) et globales (la troisième). Un réseau de contraintes à n variables est :

- *k-cohérent* ($1 \leq k \leq n$) sous la condition nécessaire et suffisante suivante. Soit une solution partielle constituée d'une instanciation de $k-1$ variables $x_1, \ldots x_{k-1}$: ces variables satisfont à toutes les contraintes pesant sur elles. Il faut et il suffit que cette solution partielle soit extensible à une variable supplémentaire. Autrement dit, si nous choisis-

sons une k-ième variable x_k, il doit être possible de trouver parmi le domaine de x_k une valeur qui satisfasse l'ensemble des contraintes pesant sur les k variables ainsi réunies.

- *fortement k_ cohérent* ($1 \leq k \leq n$) si et seulement s'il est j-cohérent $\forall j \leq k$
- *globalement cohérent* si et seulement s'il est n-cohérent

Dans le cas des STP binaires, il existe une concept équivalent à la 2-cohérence : c'est *l'arc-cohérence* d'un réseau. Cela veut dire que le réseau de contraintes ne contient aucune valeur x_i d'une variable X_i ne vérifiant pas toutes les contraintes liant X_i.

5.2 Types de cohérence

Il existe 4 types de cohérence, liées aux caractères qualitatif, quantitatif, causal et indéterministe.

5.2.1 Cohérence qualitative

La cohérence qualitative ne comprend pas de contraintes numériques, mais seulement des contraintes de structure. Ce sont les contraintes du style "avant", "après", etc. Un réseau peut être rendu incohérent par exemple par l'insertion d'une relation entre deux médias provoquant un cycle dans le graphe de contraintes. Intuitivement, cette incohérence correspond au fait que le temps s'écoule linéairement, et que nous ne pouvons revenir dans le passé.

Prenons scénario comprenant trois médias A, B et C ainsi que les relations "A before B" et "B before C". Si nous ajoutons la relation "C before A", nous avons un problème : le temps n'est pas aussi cyclique que le graphe de ces trois relations !

Madison dispose d'un mécanisme qui empêche l'auteur d'introduire une relation qui engendrerait un réseau qualitativement incohérent. Cette vérification peut être réalisée a priori, via un tri topologique du graphe ainsi qu'un algorithme de détection de circuit. Nous n'y reviendrons donc plus.

5.2.2 Cohérence quantitative

La cohérence quantitative concerne les contraintes à caractère numérique. C'est à celle-ci que nous nous intéressons le plus. L'algorithme que Madison utilise pour vérifier cette cohérence est PC2, que nous le détaillerons dans le chapitre suivant.

Il existe un autre algorithme pouvant s'acquitter de cette tâche. Il s'agit d'une version incrémentale de l'algorithme DPC (Directional Path Consistency). Il utilise le tri topologique du graphe et une forme restreinte de cohérence sur ce graphe : la DPC-cohérence. Définissons cette nouvelle cohérence. Pour cela, nous utiliserons $\otimes$, l'opération de composition d'intervalles définie selon la relation suivante :

$$[a, b] \otimes [c, d] = [a + c, b + d]$$

Soit un graphe $G = (X, A, E)$ représentant un réseau de contraintes numériques, trié topologiquement par une fonction $rang$. Notons $C_{i,j}$ l'intervalle de validité associé à l'arc joignant i à j. Le réseau G est *DPC-cohérent* si la condition suivante est vérifiée :

$$\forall i, j, k \in X \begin{cases} rang(i) < rang(k) \\ rang(j) < rang(k) \end{cases} \Rightarrow C_{i,j} \subseteq C_{i,k} \otimes C_{k,j}$$

De façon un peu plus intuitive, cela signifie que si deux instants i et j sont situés avant un autre instant k, l'intervalle contraignant le temps entre i et j est plus précis (plus petit) que l'intervalle contraignant le temps entre i et j en passant par tout autre instant k. Cela veut dire que toute contrainte entre deux instants est plus précise que la même contrainte en passant par un autre instant : la contrainte est en quelque sorte minimale.

Nous allons présenter sommairement le principe de fonctionnement de cet algorithme. Les détails peuvent être trouvés dans [Layaïda97, p. 120–124]. Considérons la figure suivante, représentant 7 médias A, B, C, D, E, X et Y. Les numéros dans les nœuds du graphe représentent l'ordre topologique sur lequel repose l'algorithme. Supposons que Y soit l'objet venant d'être inséré. Après réexamen éventuel du tri topologique, l'algorithme va étudier l'incidence du nouvel élément Y sur la DPC-cohérence, pour la rétablir. Ceci l'amènera à préciser plus l'intervalle de validité de X. La précision plus forte de X amènera celle de A. Le principe est celui d'une propagation de contraintes, celle qui assure la DPC-cohérence du réseau. Lorsque l'algorithme termine, le graphe entier est DPC-cohérent.

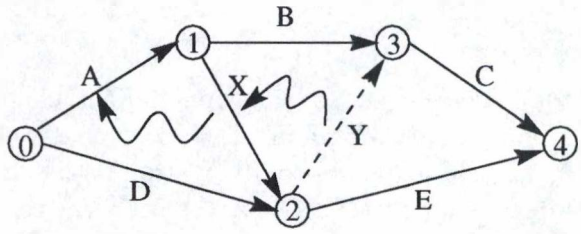


FIG. 5.1 – *Algorithme DPC-incrémental : principe illustré*

Par rapport à l'algorithme PC2 que nous présentons dans le chapitre suivant, cet algorithme a deux avantages :

- Incrémentalité. Tout comme PC 2, cet algorithme propage les contraintes de façon incrémentale sur le graphe, en exploitant la localité des modifications
- Performance. Sur ce plan, PC 2 est battu. Ce dernier traîne sa complexité cubique $O(n^3)$, tandis que l'algorithme DPC-incrémental est linéaire $O(n.k^2)$ où k est le degré moyen d'un sommet du graphe.

5.2.3 Cohérence causale

La cohérence causale est d'un autre ordre que les précédentes. Il s'agit de vérifier la cohérence d'un graphe ou d'un réseau de contraintes comportant des relations causales. Nous les avons déjà définies plus haut dans le texte, à la fin du chapitre consacré à Madeus. Comme promis à ce moment, voici l'interprétation graphique des relations causales :

Ceci illustre la difficulté. En effet, la signification précise de ces relations varie en fonction de la longueur relative des médias qu'elle concerne. La traduction des relations d'intervalles en relation d'instants ne posait aucune difficulté tant que nous nous limitions à RIA. Mais la situation est plus compliquée ici. Dans la traduction en intervalles d'instants, à la base de la construction du graphe temporel, nous n'avons à faire qu'au début et à la fin des médias, notés a^- et a^+ pour un intervalle ou média a. Ici, il faudrait distinguer deux types d'instants de fin : la fin normale, et la fin effective. En effet, un média peut être interrompu avant sa fin normale. C'est cet instant que nous appelons fin effective.

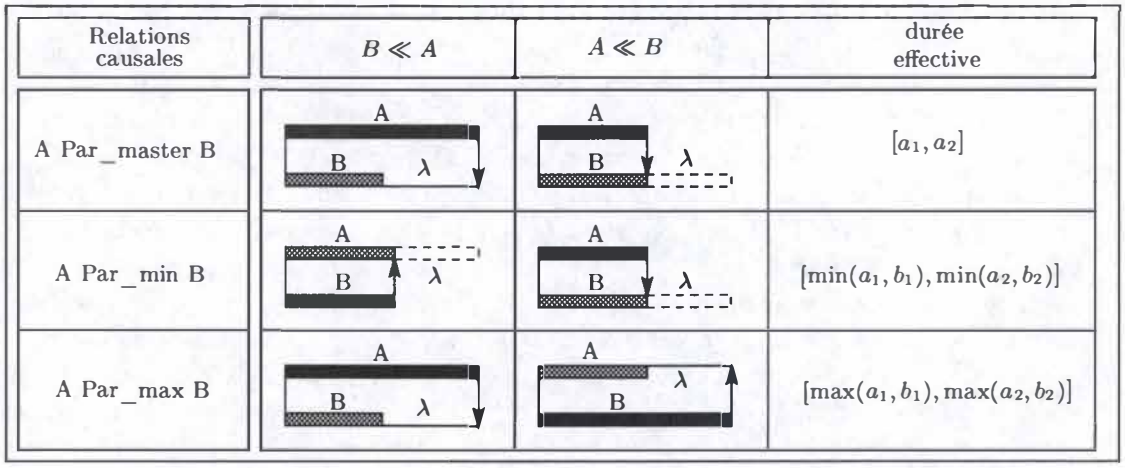


FIG. 5.2 – *Les relations causales entre* $A : [a_1, a_2]$ *et* $B : [b_1, b_2]$, *sous forme graphique*

Ce problème est loin d'être résolu, et aucun éditeur actuel ne propose vraiment de solution, si ce n'est les éditeurs à base de programmation, sortant des objectifs fixés par Madison. La construction du graphe, rendue possible par la restriction des contraintes aux STP, est mise à mal : comment gérer les deux types de fin d'un élément ? Une solution possible est d'ajouter des informations supplémentaires dans le graphe, à destination du scheduler. Ceci n'est peut-être pas la meilleure façon de faire : il faudra étudier ce problème plus en profondeur, peut-être dans un autre mémoire.

5.2.4 Cohérence indéterministe (contrôlabilité)

La cohérence indéterministe est la plus difficile à manipuler. Nous avons déjà parlé du problème de l'indéterminisme. Pour le gérer, il est indispensable d'introduire dans l'algorithme les notions d'intervalles contrôlables et incontrôlables :

- Un *intervalle contrôlable* $A : [m, \lambda M]$ est un intervalle dont la contrainte numérique de durée, son intervalle de validité $[m, M]^c$, est telle que le système d'édition peut réduire arbitrairement sa valeur entre m et M. La différence $M - m$ caractérise la souplesse que l'auteur attribue au scénario. C'est la notion de *flexibilité*.
- Un *intervalle incontrôlable* $B : [m, \lambda, M]$ est un intervalle dont la contrainte numérique de durée, son intervalle de validité $[m, M]^i$, n'est connue précisément qu'au cours de l'exécution du document. La capacité du scénario à être adapté dynamiquement lors de son exécution, par exemple en empruntant de la flexibilité ailleurs, est connue sous le terme de *contrôlabilité*.

Il est clair ici qu'aucun formatage statique n'est encore possible. Nous sommes ici face à une difficulté énorme. Nous sortons en effet largement du cadre simple et classique des STP. Il faut encore trouver des solutions générales. Pour l'instant, nous ne pouvons tout au plus que proposer des solutions partielles, couvrant quelques cas bien identifiés.

5.3 Vérification de cohérence

Nous avons défini la cohérence, déclinée selon ses types. Nous allons pouvoir enfin parler de la vérification de cohérence. Nous allons aborder ses exigences, deux algorithmes résolvant le problème (PC2 et DPC incrémental), ainsi que quelques extensions futures à apporter.

5.3.1 Exigences

Commençons par les exigences posées sur la vérification de cohérence. Celle-ci recouvre plusieurs aspects. Selon la priorité que l'on accorde à ces aspects, l'une ou l'autre méthode sera préférable. Les aspects intéressants sont surtout :

1. la *rapidité* de la vérification de cohérence, d'autant plus cruciale si elle est effectuée souvent.
2. la possibilité de vérifier la cohérence selon des niveaux *hiérarchiques*, indispensable si l'on veut éditer de gros documents
3. la possibilité de calculer les *domaines minimaux*. Ceci donne à l'auteur une idée des proportions selon lesquelles il peut modifier un document pour le garder cohérent, sans devoir relancer tout le calcul.

Dans l'optique de Madison, les 3 aspects cités ci-dessus sont importants :

1. L'exigence la plus forte et la plus difficile à remplir, c'est que les algorithmes doivent être très rapides. Nous nous situons dans un processus incrémental d'édition, un cycle "éditer, tester". La cohérence devra être vérifiée à chaque opération d'édition du document. De plus, une vue spéciale style "ligne du temps" permettra à l'utilisateur d'effectuer ces opérations. Ceci pose de sévères exigences quant à l'efficacité des algorithmes employés.
2. L'aspect hiérarchique des documents est fondamental dans Madison. Il doit pouvoir gérer efficacement de gros documents. Ce choix le différencie d'autres approches d'édition.
3. Disposer des domaines minimaux est une information précieuse pour Madison, qui a dans ses objectifs de fournir un environnement interactif intuitif pour l'auteur. Personne (ou presque) ne raisonne en termes de contraintes. Les informaticiens ayant fait de la recherche opérationnelle avec le langage OMP confirmeront deux choses. D'abord, transformer sans erreur une idée sous forme de système de contraintes n'est pas évident. Ensuite, deviner à quoi ressemblera une solution à partir d'un système de contraintes même minuscule est très difficile. C'est pourquoi Madison présente une vue style "ligne du temps". Cette métaphore a fait le succès commercial de logiciels d'édition multimédia comme Director. Cette vue permet de modifier le placement des médias d'une solution. Mais le déplacement de ces objets est limité par Madison aux mouvements permis, càd ceux se cantonnant aux domaines minimaux.

5.3.2 Exigence de rapidité : compromis

Précisons l'exigence de rapidité donnée ci-dessus. Nous avons déjà dit que la recherche de solutions dans un CSP est NP-complet, et que l'expressivité du langage de spécification de contraintes ne doit pas être trop réduite. Il existe un compromis entre rapidité des algorithmes et puissance expressive du langage. C'est ce qu'a choisi Madison.

1. La rapidité peut être atteinte en utilisant des contraintes binaires sous forme de STP. Dans un STP, les contraintes disjonctives sont interdites : toutes les contraintes sont

spécifiées sous forme d'intervalles de validité. Dans le domaine de l'édition multimédia, cela ne semble pas poser problème : il suffit de modéliser temporellement tout média par un intervalle de durée admissibles. C'est le fameux triplet $[m, \lambda, M]$. Il est évident que les algorithmes du type "backtracking" (avec retour arrière) sont à proscrire. En effet, si l'on note d la taille du plus grand domaine des variables contraintes et n le nombre de ces contraintes, l'espace de recherche sera borné par d^n. Cela donne un algorithme en temps exponentiel, ce qui n'est guère souhaitable.

2. La puissance expressive. Les contraintes à base d'intervalles (Allen) sont les plus générales. Mais nombre d'entre elles ne sont pas utiles. De plus, cela génère des algorithmes trop lents et même incomplets (certaines incohérences sont indétectables). Les relations d'instants ne sont pas suffisamment expressives. Par contre, leur disjonctions permettent de générer un sous-ensemble des relations disjonctives d'intervalles : la RIA (Restricted Interval Algebra). Celles-ci couvrent suffisamment de cas pour être un bon compromis.

5.3.3 Rapide état de l'art et choix

Nous connaissons les exigences à respecter pour vérifier la cohérence. Nous allons pouvoir examiner les algorithmes qui effectuent cette vérification.

Nous avons déjà un peu évoqué ce point plus haut, dans le chapitre consacré aux contraintes. Deux grandes classes d'algorithmes se distinguent : la programmation linéaire et les algorithmes utilisant une structure de graphes. Nous avons mentionné que la première classe pose des problèmes de performances. Ceci étant primordial pour Madison, nous ne nous y attarderons pas d'avantage et nous limiterons aux algorithmiques de graphe.

Parmi ceux-ci, nous pouvons les classer en fonctions des 3 critères définis plus haut : la rapidité, le calcul on non des domaines minimaux et la gestion de la hiérarchie. Les deux algorithmes que nous avons présentés ont les caractéristiques suivantes :

- *PC 2*. C'est un algorithme incrémental en $O(n^3)$. Il calcule et maintient les domaines minimaux en plus de vérifier la cohérence quantitative. De plus, il a été étendu pour tenir compte de la hiérarchie.
- *DPC-incrémental*. Cet algorithme incrémental est en $O(k.n^2)$. Il ne maintient pas les domaines minimaux, mais vérifie rapidement la cohérence quantitative d'un graphe entier.

Madison a porté son choix sur PC 2, car la maintenance des domaines minimaux lui est chère. Il permet un formatage efficace sans retour arrière. De plus, ces domaines fournissent à l'auteur des renseignements précieux : sa marge de manœuvre lors de la modification d'un élément. Néanmoins, pour des vérifications rapides de la cohérence, surtout au chargement d'un gros document, DPC-incrémental pourrait se révéler comme meilleure solution.

5.3.4 Extensions

Nous avons décrit la vérification de cohérence. Il nous reste à donner quelques idées d'extensions.

La vérification de cohérence quantitative n'est pas suffisante vis-à-vis des ambitions de Madison. Il faut pouvoir gérer les contraintes liées aux relations causales (interruptions). Sans cela, les possibilités d'interaction offertes au lecteur d'un document multimédia structuré seraient réduites à une peau de chagrin, style magnétoscope.

Un second défi à relever concerne la gestion des intervalles incontrôlables. La vérification de cohérence devra utiliser une autre définition de la cohérence. En effet, toutes les durées des intervalles de validité ont été considérées à présent comme connues statiquement à l'avance. Les intervalles incontrôlables changent cet état de fait. Ils exigent un formatage dynamique. La vérification de cohérence devra s'assurer que la phase de formatage ne se retrouve jamais dans des situations impossibles. Il faudra donc une cohérence plus large, qui assure la présence d'une solution lors de la présentation, et cela quelles que soient les valeurs prises par les intervalles incontrôlables. Ceci est un problème complexe sujet à recherche. Nous tenterons de l'aborder au chapitre 8.

5.4 Formatage

La vérification de cohérence concernait l'édition de document. Penchons-nous un instant sur leur présentation : le formatage. Celui-ci sera réalisé en réutilisant l'algorithme PC2 qui vérifie la cohérence. Nous allons adopter la même structure que pour la vérification de cohérence : présenter les exigences, les algorithmes, puis les extensions à apporter. Nous profiterons de l'occasion pour apporter une précision souvent mal comprise, concernant la sémantique de la cohérence.

Le formatage consiste à trouver une solution, un ordonnancement du document respectant toutes les contraintes (spatio-)temporelles. Le nombre de solutions est rarement unique. Si tel était le cas, l'approche par contraintes n'aurait aucun avantage par rapport à un style d'édition à la Director, qui présente toujours une seule "solution".

Idéalement, il faut que l'auteur puisse naviguer parmi l'ensemble des solutions possibles. C'est le but de la vue "scénario" que présente l'interface de Madison. Il s'agit d'une vue comportant une ligne du temps. Elle permet l'édition directe de caractéristiques des médias. De plus, lors de la modification du placement temporel d'un élément, une ligne rouge délimite les placements possibles qui ne sortent pas d'une solution. Cette possibilité apparaît comme fondamentale pour amener des auteurs à utiliser l'approche par contraintes, pas très naturelle a priori. Ce type de visualisation est étudié dans d'autres travaux, comme [Tardif97] et [Meurisse99].

5.4.1 Précisions concernant aussi la cohérence

Le formatage soulève un problème d'interprétation du terme "cohérence". Il est souvent mal compris. C'est pourquoi nous insistons ici sur sa sémantique, en donnant un exemple concret de mauvaise interprétation.

Lors de la phase de formatage (calcul d'une solution), disposer des domaines minimaux permet de choisir l'instanciation d'une variable sans risque de remise en question plus tard de ce choix. Ceci demande une petite précision. Un aspect souvent mal compris dans la cohérence est le choix d'instanciation des variables qu'elle permet. Disposer d'un domaine minimal ne signifie pas pouvoir choisir n'importe quelle valeur parmi chaque intervalle de validité. Rappelons-nous que le type de cohérence assurée par PC 2 est la 2 -cohérence. L'exemple suivant va nous éclairer.

Soit un graphe (de contrainte) minimal tel que celui de la figure 5.3. Choisissons 8 comme durée pour A et 16 comme durée pour B. Ce faisant, nous rendons le graphe incohérent, car le dernier choix pour C est imposé. C doit durer 8 unités, alors que son intervalle de validité est de $[10, 20]$. Les graphes minimaux remettraient-ils des choix en question ?

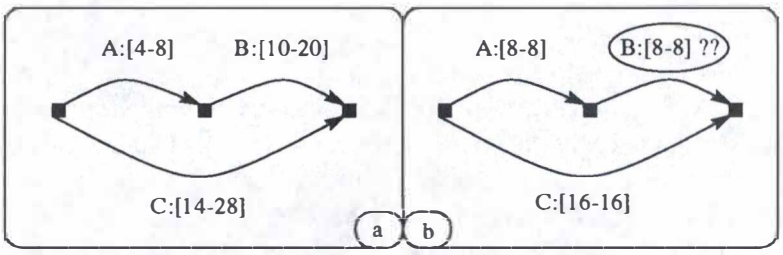


FIG. 5.3 – *Comprendre la cohérence. Le domaine minimal de cohérence (a) ne veut pas dire suite de choix sans revérification de cohérence : après choix des valeurs 8 et 16, le graphe devient incohérent (b) !*

Et bien, la réponse est bien sûr : non. Il suffit de mieux comprendre comment effectuer le choix des instanciations. Elles ne seront jamais remises en question. Simplement, chaque fois qu'un intervalle de durée est choisi, il faut relancer la vérification de cohérence, qui calculera les nouveaux domaines minimaux. En effet, fixer la durée d'un média entraîne généralement une plus grande précision des autres intervalles de validité. Ceux-ci devenant plus petits (plus précis), il peut arriver qu'ils soient réduits à une unité, auquel cas la durée en est fixée, ou bien ils se réduisent à rien, auquel cas une incohérence est détectée. Nous obtenons alors un nouveau graphe minimal. La valeur qui vient d'être choisie ne sera plus modifiée. Il suffit ensuite de recommencer le processus précédent : choisir une valeur, recalculer le graphe minimal et ainsi de suite.

Voyons cela sur l'exemple précédent, à la figure 5.4. Nous commençons par choisir 16 comme valeur de C. Mais cette fois, nous n'oublions pas de recalculer le graphe minimal ('b' sur la figure). Nous verrons plus loin comment ce calcul a été effectué, dans le chapitre consacré à PC2. Admettons le résultat pour le moment. Nous voyons cette fois que nous ne pouvons plus choisir 8 comme valeur pour A. En effet, son intervalle de validité $[4, 6]$ ne contient plus la valeur 8. Le nouveau graphe minimal illustre bien la notion de 2-cohérence que PC2 utilise : nous pouvons choisir n'importe quelle valeur pour A (resp. B), et il sera toujours possible d'en trouver une pour B (resp. A) qui vérifie toutes les contraintes. Le choix pour une des 2 variables restantes forcera le choix de la troisième, par recalcul du graphe minimal. Après ce choix, le graphe contiendra presque toujours des intervalles réduits à un point (une seule valeur). Par exemple, choisir 4 pour A force le choix de B à 12 ('c' sur la figure). Choisir 6 pour A force le choix de B à 10 ('d' sur la figure). Ce sont les deux choix extrêmes pour l'intervalle A. Nous pouvons en supposer la validité pour les valeurs non extrêmes. Le même genre de conclusion peut être tiré en partant de B.

5.4.2 Exigences

La vue scénario de Madison pose une nouvelle fois des exigences encore plus sévères quant à la vitesse d'exécution des algorithmes. Les autres exigences, moins importantes, sont du même ordre que celles vues pour la cohérence (section 5.3.1 page 71).

5.4.3 Rapide état de l'art et choix

La situation est sensiblement la même que celle pour la vérification de cohérence. La programmation linéaire est trop lourde dans notre contexte. En ce qui concerne les algorithmes

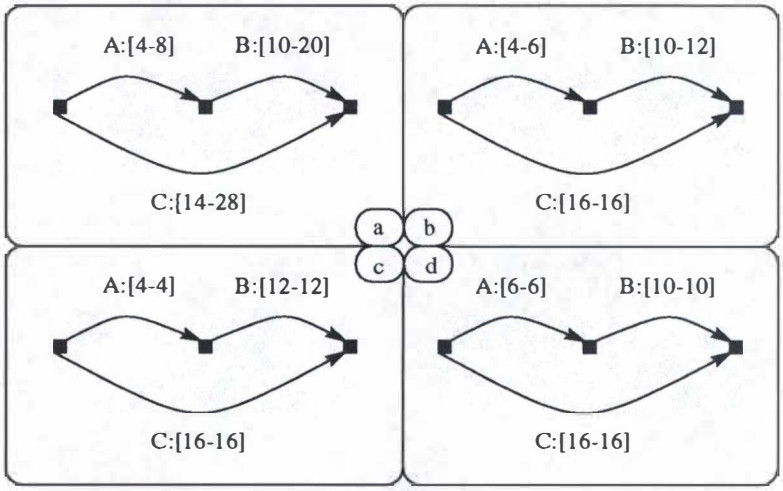


FIG. 5.4 – *Comprendre la cohérence. Partant du graphe minimal (a), nous choisissons 16 comme valeur de C. Cela conduit à un nouveau graphe minimal (b). Ne pouvant plus choisir 8 pour A, nous choisissons par exemple 4 (en c) ou 6 (en d).*

de graphe, DPC ne maintient pas les domaines minimaux: le choix d'instanciation d'une variable pourrait être remis en cause plus loin dans le processus, obligeant à des retours en arrière coûteux ("backtracking"). C'est donc PC 2 qui sera utilisé.

L'avantage principal lors du calcul d'une solution est que l'instanciation d'une variable peut se faire sans souci de remise en question plus tard. L'algorithme sera donc des plus efficaces.

5.4.4 Extensions

Les extensions à apporter sont du même ordre que celles présentées pour la cohérence: gérer les relations causales et les intervalles incontrôlables. Il faudra un formatage dynamique, au sens où certaines durées devront être calculées en fonction d'éléments connus seulement au cours de la présentation proprement dite. Le scheduler dynamique devra être doté d'une intelligence bien supérieure à celle du scheduler statique.

Mais avant cela, il nous faudra détailler tout de même PC2, cet algorithme au cœur de mon travail lors du stage à l'INRIA de Grenoble. Il faut également précicer que l'INRIA Rhône-Alpes se situe en réalité à une quinzaine de kilomètre de Grenoble, dans une petite bourgade répondant au doux nom de "Montbonnot St-Martin" [1].

1. Ceci valait la peine d'être dit, non?

Chapitre 6

L'algorithme classique PC2

Nous avons bien délimité le pourtour des questions abordées dans ce mémoire. Nous avons choisi un éditeur de documents multimédia basé sur les systèmes de contraintes temporels. Il s'appelle Madison. Nous savons qu'il faut déterminser si le système possède ou non une solution, càd qu'il faut en vérifier la cohérence. PC2 est un algorithme qui s'acquitte de cette tâche. C'est lui que nous décrivons dans ce chapitre.

Lors de mon séjour à l'INRIA, j'ai pu implémenter l'algorithme classique PC2 (de l'anglais "Path Consistency II"), ainsi qu'une adaptation de ce dernier aux scénarios hiérarchiques. Le principe à la base de PC2 n'étant pas si trivial, il est décrit plus précisément dans ce qui suit. Dans le chapitre suivant, nous détaillerons les aspects de l'implémentation réalisée à Grenoble, ainsi que les difficultés rencontrées.

Dans ce chapitre, nous décrivons le principe d'un algorithme. Il est logique de commencer par décrire la structure de données qu'il emploie : un graphe. PC2 manipule directement les intervalles de validité des médias. Il lui faudra disposer d'opérations sur les intervalles. Elles seront définies dans la section 6.2 page 78. Nous serons à même alors de décrire le principe utilisé par PC2, à savoir une propagation de contrainte. Cela sera fait à la section 6.3 page 79. Par la suite, nous pourrons donner l'algorithme en pseudo-code (section 6.4.1 page 81). Cet algorithme n'étant pas trop simple, nous l'examinerons pas à pas, à la lumière d'un exemple. Puis nous donnerons ses postconditions, ainsi qu'une preuve de sa terminaison.

6.1 Graphe et structure de données

Voyons pour commencer la structure de données sur laquelle s'appuie l'algorithme PC2. Il utilise une structure de graphe orienté (ensembles de sommets et d'arcs les joignant). Les sommets représentent des instants temporels. Un arc i, j joignant les sommets i à j représente un objet à jouer : soit un média, soit un délai à respecter. Il signifie qu'une fois arrivé au temps i, l'objet est présenté pendant un temps t, et est terminé au temps j. Il est possible que le temps t soit négatif pour un arc i, j. Dans ce cas, l'instant j précède l'instant i de $-t$ unités de présentation, et l'objet a une durée de t unités. Chaque arc (objet ou délai) est étiqueté par un triplet $[m, \lambda, M]$. L'objet représenté est soumis à cette contrainte : lorsqu'il sera joué en un temps t, il devra l'être dans cet intervalle (on veut t tel que $m \leq t \leq M$). Ce triplet représente les durées minimales (m), maximales (M) et préférées (λ) souhaitées pour l'exécution de l'objet. Ceci permet une bonne flexibilité dans la spécification d'un scénario : l'auteur n'es pas forcé de donner une durée précise pour les médias qu'il veut présenter. Notons

que PC2 ne tient aucun compte de la valeur préférentielle λ au cours de ses calculs. Dans la suite du texte, nous omettrons donc cette valeur.

Pour assurer sa terminaison, PC2 a besoin en entrée d'un graphe orienté *complet*, au sens mathématique du terme (ou presque) : tous les sommets doivent être reliés deux à deux par un arc les joignant. Si le graphe n'est pas complet au départ, il est complété par des arcs à durée indéterminée de type $[-\infty, +\infty]$. En effet, l'intervalle doit comprendre des durées négatives, car on ne peut pas connaître a priori lequel des deux instants i et j précédera l'autre à l'exécution. Notons qu'il n'est pas nécessaire que le graphe soit complet pour chaque couple de nœuds "dans les deux sens". En effet, si un arc i, j de durée $[m, M]$ existe déjà, il est inutile d'ajouter l'arc j, i : sa durée, calculable, est de $[-M, -m]$, car

$$m \leq t \leq M \Leftrightarrow -m \geq -t \geq -M \Leftrightarrow -M \leq -t \leq -m$$

Étant donnés tous les intervalles de durée $[m, M]$ pour tous les arcs i, j du graphe complet en entrée, l'algorithme doit vérifier si ces durées sont cohérentes. PC2 vérifie un type particulier de cohérence, définie au chapitre précédent : la 2-cohérence. Il utilise une technique de propagation des contraintes : cela lui permet également de calculer le graphe minimal. Mais pour cela, il doit disposer d'opérations sur les intervalles de durées. Les deux opérations de composition ($\otimes$) et d'intersection ($\oplus$) suffiront.

6.2 Opérations sur les intervalles

Nous avons une structure de données. Via cette structure de graphe, PC2 va manipuler directement les intervalles de validité étiquetant les arcs. Il lui faut donc qu'il dispose d'opérations sur les intervalles. Les deux opérations de composition $\otimes$et d'intersection $\oplus$suffiront. Les voici.

6.2.1 La composition ($\otimes$)

Commençons par définir l'opération de composition d'intervalles, notée $\otimes$. Etant donnés donnés deux intervalles de durée $[m_1, M_1]$ et $[m_2, M_2]$ représentant deux objets O_1, O_2 consécutifs, donner l'intervalle de durée qui représenterait un seul objet O de durée équivalente à la séquence O_1, O_2. D'après la figure 6.1, on a:

$$[m_1, M_1] \otimes [m_2, M_2] = [m_1 + m_2,\ M_1 + M_2]$$

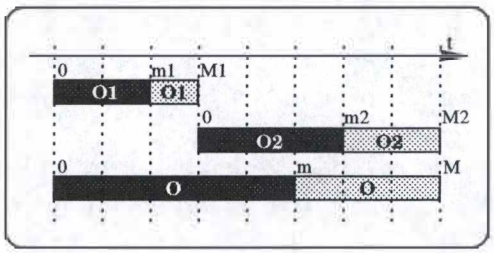


FIG. 6.1 – *Composition ($\otimes$) d'intervalles*

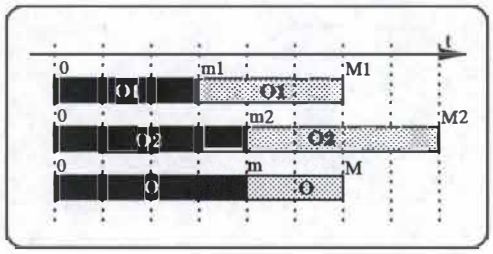


FIG. 6.2 – *Intersection ($\oplus$) d'intervalles*

6.2.2 L'intersection ($\oplus$)

Continuons en définissant l'opération d'intersection entre intervalles, notée $\oplus$. Etant donnés deux intervalles de durée $[m_1, M_1]$ et $[m_2, M_2]$ représentant deux objets O_1, O_2, il faut calculer l'intervalle de durée qui représenterait un seul objet O, de durée équivalente et compatible avec les durées des deux objets O_1, O_2. D'après la figure 6.2, on a:

$$[m_1, M_1] \oplus [m_2, M_2] = [\max(m_1, m_2),\ \min(M_1, M_2)]$$

6.3 Propagation de contrainte

Nous voila muni de deux opérations sur les intervalles. Nous allons pouvoir décrire précisément la technique utilisée par PC2: la propagation de contraintes sur les intervalles de validité du graphe. Pour cela, il faut définir une opération appelée réduction. Nous la définissons ci-dessous. Ce n'est qu'après que nous pourrons détailler le principe.

6.3.1 Réduction

Afin de calculer les domaines minimaux, PC2 va combiner des intervalles entre eux afin de réduire le plus possible les intervalles de validité. Appelons "réduction" cette opération qui consiste à réduire l'intervalle $[m, M]$ d'un arc en le combinant avec deux autres arcs. Définissons-la de manière plus précise.

Pour procéder à une réduction, PC2 choisit un arc i, j et un sommet k. Il peut alors examiner l'influence de l'arc i, j sur l'arc i, k puis sur l'arc k, j. C'est ce qu'illustre la figure 6.3. Une réduction examine donc un triplet d'arcs formant un triangle dans le graphe orienté complet.

Étudions d'abord l'influence de l'arc i, j sur l'arc i, k. Pour la simplicité, nous noterons les arcs par leurs sommets, séparés par une virgule (comme l'arc i, j). Nous noterons les intervalles de durée étiquetant un arc par les lettres de leurs sommets (comme l'intervalle ik). Intuitivement, passer directement de i à j est plus rapide que de passer de i à j puis de j à k. En réalité, ces deux chemins mènent au même instant. Donc, il faut qu'il y ait égalité entre l'intervalle ik et l'intervalle composé $ij \otimes jk$. Dans le triangle tel que l'auteur l'a proposé, ce n'est pas forcément le cas. Nous allons donc prendre l'intersection entre l'intervalle ik et l'intervalle $ij \otimes jk$. Nous appellerons ik' cette intersection. Si l'intersection ik' est vide, c'est qu'il y a incohérence. Sinon, nous savons que ik' sera un intervalle plus précis que ik ($ik' \subseteq ik$). En effet, un ensemble est contenu dans son intersection avec un autre: $A \subseteq (A \cap B)\ \forall A, B$.

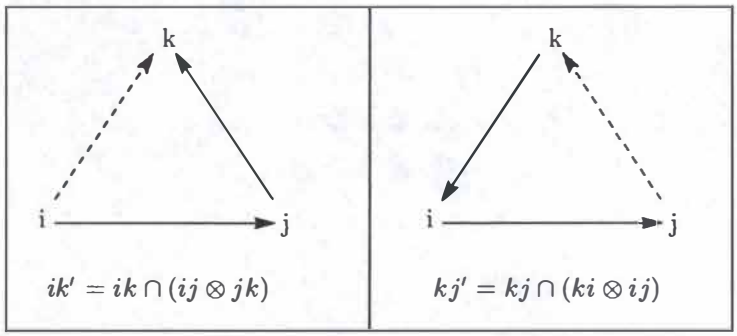


FIG. 6.3 – *Opération de réduction : influence de l'arc ij sur les arcs ik et kj*

Nous allons donc préciser ik en lui donnant comme nouvelle valeur ik'. Cela correspond à préciser la contrainte pesant sur l'arc i, k. En bref, la réduction de ik avec ij et jk est :

$$ik' = ik \cap (ij \otimes jk)$$

De la même manière, l'influence de l'arc i, j sur l'arc k, j est étudié par une nouvelle valeur kj' de l'arc k, j. Elle vaut :

$$kj' = kj \cap (ki \otimes ij)$$

En bref, une opération de réduction consiste simplement à considérer un triangle de sommets, à calculer ik' et kj', puis à en tirer les conclusions. Soit l'un de ces deux nouveaux intervalles est vide, auquel cas une incohérence est détectée, soit aucune ne l'est. Dans ce cas, la réduction est opérée en affectant aux arcs i, k et k, j leurs nouvelles valeurs, plus précises : ik' et kj'.

6.3.2 Propagation

Voilà enfin la description du principe utilisé par PC2 : la propagation des contraintes, relativement simple. D'abord, il faut choisir un arc i, j, pour étudier son influence sur tous les autres. Tour à tour, choisissons un sommet $k \neq i, j$. Après calcul des réductions ik' et kj', nous affectons ces nouvelles valeurs d'intervalles aux arcs i, k et k, j (sauf si l'un des deux est vide, auquel cas une incohérence est détectée). Une fois tous les sommets k utilisés, nous avons réduit les contraintes de tous les arcs ayant i ou j comme origine ou extrémité. Nous avons étudié tous les triangles ayant l'arc i, j comme base. Il nous reste à choisir un autre arc i, j. Mais comment faire ?

La réponse est simple. L'algorithme s'arrête s'il découvre une incohérence. S'il n'en découvre pas, il opère des réductions. Une réduction ik' sur ik est plus précise ou égale que ik. Si elle est égale, il ne sert à rien de modifier i, k. Par contre, si $ik' \neq ik$, il faut modifier l'arc i, k et étudier son influence sur les autres, plus tard. C'est ce en quoi consiste la propagation. Ceci se réalise en ajoutant l'arc i, k dans une pile d'arcs dont il faut examiner l'influence. Voilà la marche à suivre. Nous allons donner l'algorithme utilisé. Ensuite, nous l'appliquerons sur un petit exemple.

6.4 Algorithme

Nous avons étudié les principes et les concepts intervenant dans PC2. Nous sommes désormais en mesure de donner son pseudo-code. Nous l'illustrerons sur un exemple, puis nous donnerons les postconditions et une preuve de terminaison.

6.4.1 Pseudo-code

Donnons maintenant le pseudo-code de PC2. Soit P une pile destinée à contenir les arcs dont il faut étudier l'influence sur les autres. Soit également *depil* la fonction enlevant un arc au sommet de la pile P et $empil(a)$ la fonction qui ajoute un arc a à la pile P.

La phase d'initialisation de l'algorithme n'a guère été abordée. Cette phase consiste à mettre dans la pile P les intervalles dont il faut étudier l'influence sur les autres. Dans le cas d'une vérification de cohérence, il faut mettre tous les arcs du graphe dans la pile. Dans le cas d'un formatage où l'on vient de choisir la durée d'un intervalle, il suffit de mettre dans la pile P l'arc ayant cet intervalle pour étiquette. La pile peut évidemment contenir plusieurs arcs au départ. Nous voici parés pour donner enfin le pseudo-code de l'algorithme PC2 (sans extension à la hiérarchie) :

Initialisation

$empil(a_i)$ $\forall a_i$, où les a_i sont les arcs initiaux choisis. Dans le cas d'une vérification de cohérence, il faut choisir *tous* les arcs du graphe à vérifier.

Propagation

Tant que $Q \neq \emptyset$

1. Considérer un arc : $i, j \leftarrow depil(P)$
2. $\forall k \in \{1..n\}, k \neq i, j$
   - Calcul de l'influence de l'arc i, j sur les arcs i, k et k, j

$$\begin{cases} ik' = ik \cap (ij \otimes jk) \\ kj' = kj \cap (ki \otimes ij) \end{cases}$$

   - Détection d'incohérence si ik' ou kj' est vide
   - Propagation si nécessaire

     Si $ik' \neq ik$ Alors $ik \leftarrow ik'$ et empil(i, k)

     Si $kj' \neq kj$ Alors $kj \leftarrow kj'$ et empil(k, j)

6.4.2 Un exemple

Illustrons ceci sur un exemple, pour mieux comprendre. Nous avions promis de donner une explication quant aux résultats obtenus plus haut lors d'une illustration du chapitre sur la cohérence. Il s'agissait de la figure 5.4 page 75. C'est ce que nous allons faire maintenant, en illustration de l'exécution de PC2. Commençons par le résultat à expliquer, selon la figure 6.4. Nous partons du graphe en (a) sur la figure, après avoir choisi 16 comme valeur pour le média C. PC2 est exécuté sur ce graphe et renvoie le graphe en (b). Nous allons détailler ce qui s'est passé. Cette illustration est très simple par rapport au cas général. Ici, il n'y aura vraiment qu'une seule réduction.

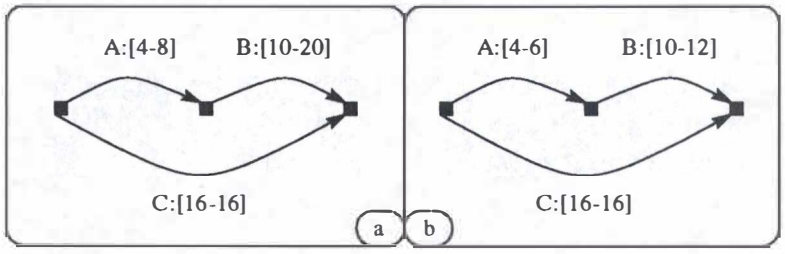


FIG. 6.4 – *Exécution de PC2 illustrée sur le graphe (a). La pile est évidemment initialisée avec l'arc* $C : [16, 16]$. *Le graphe (b) est celui renvoyé par PC2.*

Nous allons commencer par dessiner le triangle mettant en jeu les 3 médias A, B et C. Ensuite, il nous faut orienter convenablement les flèches pour étudier l'influence de C sur A, puis celle de C sur B. Ceci constituera une réduction. En nous rappelant que l'étiquette associée à l'arc inverse d'un arc d'étiquette $[m, M]$ est $[-M, -m]$, nous sommes en mesure de dessiner le triangle d'influence de C sur A avec les bons intervalles de durées. Nous avons ici un intervalle négatif : c'est normal, comme nous l'avions annoncé précédemment.

Consultons la figure 6.5 pour étudier l'influence de C sur A, ou de ij sur ik, selon les notations de l'algorithme. Nous avons bien sûr déjà enlevé l'arc i, j de la pile P, qui est vide à présent. Calculons la durée du chemin passant de i à k via j. Elle est de $ij \otimes jk$. Plus clairement : $[16, 16] \otimes [-20, -10] \stackrel{def}{=} [16-20, 16-10] = [4, 6]$. En comparant avec la durée $[4, 8]$ du chemin ik, nous constatons une différence qui ne doit pas exister : les sommets représentent des instants identiques, quel que soit le chemin parcouru. Calculons donc ik', la nouvelle durée de i, k. C'est l'intersection de $[4, 8]$ avec $[4, 6]$. Donc, $ik' = [4, 6]$. Celle-ci est plus précise que la valeur précédente ($ik = [4, 8]$). Nous affectons donc l'intervalle $[4, 6]$ à l'arc i, k, càd à l'objet A sans inversion de sens. Et nous mettons l'arc i, k dans la pile P pour propager éventuellement cette modification.

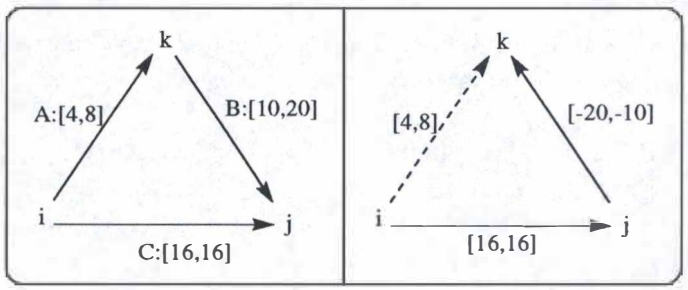


FIG. 6.5 – *Exécution de PC2 sur un exemple. Influence de C sur A : triangle avec intervalles correctement orientés*

Nous allons faire de même pour l'objet B (arc j, k avec inversion de sens), mais un peu plus vite cette fois. Le dessin des arcs orienté se trouve sur la figure 6.6. Le chemin le plus long mesure $ki \otimes ij = [-8 + 16, -4 + 16] = [8, 12]$. Par conséquent, nous obtenons la nouvelle valeur de kj, qui est $kj' = [10, 20] \cap [8, 12] = [10, 12]$. Nous empilons également k, j sur la pile. La nouvelle durée de B est $[10, 12]$.

Nous sommes déjà arrivés au résultat attendu, selon la figure 6.4 qui était à expliquer. Mais l'algorithme n'est pas terminé. La pile P contenait uniquement l'arc i, j au départ.

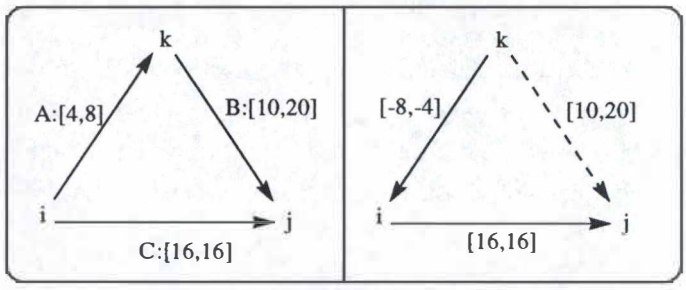


FIG. 6.6 – *Exécution de PC2 sur un exemple. Influence de C sur B : triangle avec intervalles correctement orientés*

Maintenant, il contient les arcs i,k et k,j. Continuons donc l'exécution de l'algorithme.

Étudions l'influence de l'arc A (ou i,k) sur les autres, en dépilant i,k de P. La pile ne contient plus que C (ou k,j), à présent. Cet arc va devenir notre nouvel arc i,j. Attention au changement de notations ! Pour cela, nous consultons la partie (a) de la figure 6.7. Attention au sens des flèches ! L'influence de A (ou i,j) sur i,k génère un nouvel intervalle de valeur ik' qui vaut $[16,16] \cap [4+10, 6+12] = [16,16]$. Autrement dit, cela reste inchangé : inutile d'ajouter i,k dans la pile P. L'influence de A ou i,j sur k,j génère un nouvel intervalle de valeur kj' qui vaut $[-12,-10] \cap [-16+4, -16+6] = [-12,-10]$. De nouveau, cette valeur est identique à la précédente et PC2 n'ajoute pas k,j dans la pile P.

Dépilons le dernier arc de la pile, C (ou k,j). Nous pouvons étudier l'influence de ce nouvel arc i,j (ou C) sur les autres (A et B). Nous laissons les détails en exercice. Ici non plus, les deux autres arcs ne changent pas de valeur. La pile est vide : nous avons terminé l'algorithme sans rencontrer d'incohérence. Nous avons donc obtenu un graphe minimal. C'est lui qui expliquait le résultat que nous avions dû admettre lorsque nous avons vu l'exemple de la figure 5.4 page 75.

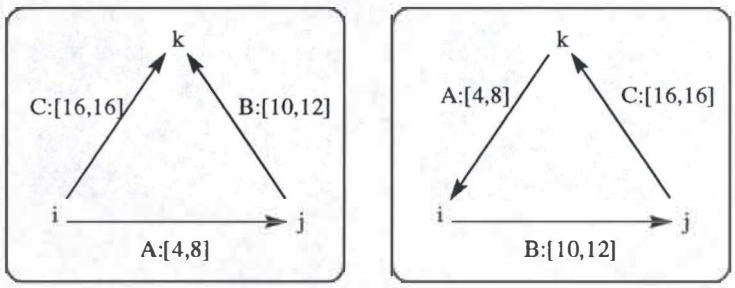


FIG. 6.7 – *Exécution de PC2 sur un exemple. Influence de A sur B et C (partie a) et influence de B sur A et C (partie b)*

6.4.3 Résultats en sortie de l'algorithme

L'algorithme PC2 est bien défini. Voici les postconditions qui seront assurées.

La propagation va continuer jusqu'à obtention du graphe minimal, dans lequel il n'est plus possible de restreindre les contraintes en les combinant avec d'autres. A moins qu'une

incohérence soit détectée, les résultats suivants sont garantis en fin de l'algorithme :

- Soit PC2 a détecté une incohérence (intersection vide) et s'arrête
- Soit PC2 déclare le graphe cohérent et renvoie son graphe minimal.
- Dans les deux cas, l'algorithme se termine correctement en un temps fini
- Son temps d'exécution est théoriquement de $0(n^3)$.

6.4.4 Terminaison

La terminaison de l'algorithme est assurée par la complétude du graphe, le nombre fini de sommets et d'arcs, et surtout la longueur finie et entière des intervalles de validité. En effet, à chaque pas de l'algorithme, la largeur des intervalles que l'on ajoute dans la pile a diminué strictement. Sinon, ces intervalles ne sont pas ajoutés à la pile. Nous sommes assurés de la terminaison, car nous avons choisi des valeurs entières pour les limites de durée d'intervalles.

La terminaison n'aurait pas été garantie si nous avions opté pour des valeurs réelles. Supposons en effet que la taille d'un intervalle décroisse selon la suite :

$$I_1 = \left[0, \frac{1}{1}\right], I_2 = \left[0, \frac{1}{2}\right], I_3 = \left[0, \frac{1}{3}\right], \ldots, I_n = \left[0, \frac{1}{n}\right]$$

Dans ce cas, il est clair que l'auteur peut encore attendre longtemps avant la fin du calcul des domaines minimaux !

6.4.5 Une note au passage

Nous venons de présenter les idées derrière l'algorithme PC2. Mais nous n'avons rien dit de la gestion de la hiérarchie, et encore moins de son intégration au sein d'une application aussi complexe que Madison. La simplicité apparente ci-dessus est exprimée hors-contexte. En effet, l'implémentation en java de ce code, dans Madison, comporte environ 1 500 lignes, sans compter, avec la structure de graphe, les quelque quatre mois de travail que cela représente ! D'ailleurs, nous allons voir cela d'un peu plus près dans le chapitre suivant, consacré à mon implémentation de PC2 à l'INRIA.

Chapitre 7

Implémentation de PC2

Nous venons de voir au chapitre précédent les principes de fonctionnement de PC2. Rappelons qu'il s'agit d'un algorithme qui propage des contraintes sur un graphe jusqu'à obtension du réseau minimal ou jusqu'à détection d'une incohérence. Mais ceci est une vue académique. Il nous faut une méthode pour l'analyse de scénarios (vérification de cohérence) et pour la synthèse des présentations (formatage spatio-temporel de documents). Or, il est reconnu que toute bonne méthode doit :

1. être fondée sur des *modèles* (théoriques). Cela a été fait dans les chapitres précédents
2. être mise en œuvre à l'aide d'*outils logiciels*. Il s'agit du vérificateur de cohérence et du formateur temporel, dans notre cas. Cette phase est en cours de réalisation à l'INRIA, où j'ai pu apporter ma contribution lors de mon stage du premier semestre de l'année académique 1998–1999. Mais ces aspects n'ont pas encore été abordés vraiment jusqu'ici. Cela sera chose faite après ce chapitre.
3. proposer une démarche formée d'étapes et de règles pour la mise en œuvre de ces modèles et outils en vue de construire et de décrire une solution détaillée. C'est en quelque sorte l'intégration des éléments précédents, au cas par exemple où plusieurs modèles sont utilisés chacun à leur niveau. Dans le cas de Madison, cette phase est si simple qu'elle en est réduite à sa plus simple expression. Je ne m'y attarderai donc plus.

En d'autres termes, même s'il n'y a rien de plus pratique qu'une bonne théorie, elle ne sert pas à grand chose si elle n'est pas utilisée dans la pratique via des outils logiciels. C'est pourquoi je m'attarde dans ce chapitre aux aspects concernant l'implémentation. De plus, les pages précédentes ont passé plusieurs aspects sous silence. Je vais donc évoquer la face cachée de l'algorithme au cœur de Madison, PC2. Ce chapitre concerne un aspect original du stage et du mémoire. C'est pourquoi j'ai utilisé et utiliserai souvent la première personne.

7.1 Structure du chapitre

Ce chapitre est structuré de la manière suivante. Les chapitres précédents ont décrit le principe général de PC2 dans sa version la plus simple. Son implémentation a l'air triviale, mais ce n'est pas vraiment le cas. Lors du passage de la théorie à la pratique, il faut soulever tous les voiles sombres, préciser ce qui ne l'a pas été, sans oublier les éventuels effets de bords. Il est fréquent d'oublier un petit détail qui peut avoir son importance. C'est ce que nombre d'informaticiens appellent des "bugs". Ces derniers sont vilains et pernicieux, car ils se glissent

sciemment dans le code, à l'insu des programmeurs . Pourtant, je me demande s'il ne s'agirait pas plutôt d'erreurs, terme moins reluisant, certes, mais plus proche de la réalité. Tout ceci pour dire qu'un comportement a priori mystérieux et inexplicable de mon implémentation de PC2 a mis au jour un oubli : la façon dont est construit le graphe temporel est primordial. En particulier, la partie qui fixe les intervalles de validité lors de la construction. C'est pourquoi je commence par décrire plus en détail la structure de graphe temporel, en section 7.2 page 86.

Le graphe temporel constitue une face cachée de PC2. Son implémentation a permis d'en déceler une autre. Le graphe utilisé par PC2 doit être complet. Bien évidemment, celui issu des spécifications de l'auteur ne l'est pas. Il fallait résoudre ce problème avant d'aller plus loin. Ce n'est encore que la base de PC2, celle qui ne considère que les documents sans hiérarchie. La section 7.3 page 89 considère les problèmes d'implémentation de cette première version de PC2.

Une fois ces problèmes réglés, il faut tenir compte de la structure hiérarchique des documents. Je présente donc l'extension de PC2 à une structure hiérarchique, sans laquelle l'auteur perd beaucoup d'intérêt à utiliser Madison. C'est la section 7.4 page 93. Le but avoué est de pouvoir traiter des documents de taille conséquente de manière suffisamment efficace. Pour cela, il faut exploiter le plus possible la localité des modifications.

Ceci met une fois de plus en évidence les problèmes de performances, que j'aborde en section 7.6 page 102. Ces problèmes étaient relégués aux oubliettes dans la partie théorique. Ils sont pourtant de taille, et le choix du langage Java n'arrange pas vraiment les choses, au contraire. Il a donc fallu contourner certaines limites et effectuer certains choix. J'ai contribué à l'édifice, mais il reste bien des choses à faire. Il faudra encore définir et implémenter de futures améliorations, pour lesquelles je donnerai quelques suggestions.

Les problèmes de performances sont encore plus aigus lorsque l'on fait souvent appel à la vérification de cohérence réalisée par PC2. C'est le cas dans l'algorithme de formatage, que je décris dans la section 7.5 page 100.

7.2 Construction du graphe temporel

La construction du graphe temporel est importante, car c'est elle qui génère les durées étiquettant les arcs. Ces durées sont bien sûr les intervalles de durée $[m, \lambda, M]$. Par abus de langage, j'emploierai parfois l'un pour l'autre. C'est cette construction de graphe qui m'a fait perdre un certain temps pour trouver la cause d'un comportement bizarre de mon implémentation. L'algorithme n'était pas en cause, mais bien la génération des durées sur les arcs. Il faut dire que mon temps à l'INRIA s'est essentiellement réparti entre le code source consacré aux graphes et celui concernant la vérification de cohérence. J'ai terminé mon stage par une implémentation aussi rapide que rudimentaire d'un système de formatage. Penchons-nous donc sur la construction du graphe. Nous nous limitons pour l'instant à un graphe non hiérarchique.

Je commence par décrire l'insertion des médias, puis de leurs relations temporelles. Cela pose le problème des durées à choisir pour les délais insérés, sans compter celui de la fusion de sommets. Ensuite, je résume les types de durées à affecter aux types d'arcs.

7.2.1 Insertion des médias

Le graphe commence par deux seuls instants : les instants de début et de fin du document. Ensuite, lors du chargement d'un document par exemple, ce graphe va s'agrandir. Il va y avoir

insertion de médias. Cette insertion est simple. Un arc est créé dans le graphe pour le nouveau média.

Ensuite, les instants de début et de fin du média sont reliés aux instants de début et de fin du document. Ceci implique la création de deux nouveaux arcs de type fictif. Ces arcs ne sont pas vraiment de type "délai". En effet, un arc de type délai ne peut contenir la valeur 0 dans son intervalle de validité. Si tel était le cas, le graphe pourrait contenir des arcs de durée 0. Autrement dit, certains instants (sommets) pourraient être reliés par un arc de durée nulle. Ceci voudrait dire que deux sommets différents représentent le même instant. Cela n'a pas de sens. Si deux instants sont identiques, ils doivent être représentés par le même sommet. Sans quoi, la sémantique du graphe, devant être revue de fond en comble, deviendrait assez bizarre, voire ingérable. Bref, les arcs fictifs liant le début du document au début du média inséré doivent comporter une durée de $[0, +\infty]$, car ils ont le privilège d'accepter 0 dans leur domaine. Par contre, ils ne peuvent pas comporter de valeur négative. En effet, nous connaissons leur orientation. Dans le cas contraire, PC2 pourrait découvrir des graphes minimaux pour le moins étranges. Le formatteur temporel pourrait recevoir des arcs de durée négatives, alors que cela n'est pas possible étant donnée son implémentation. Le non respect de cette précondition aurait des effets imprévisibles.

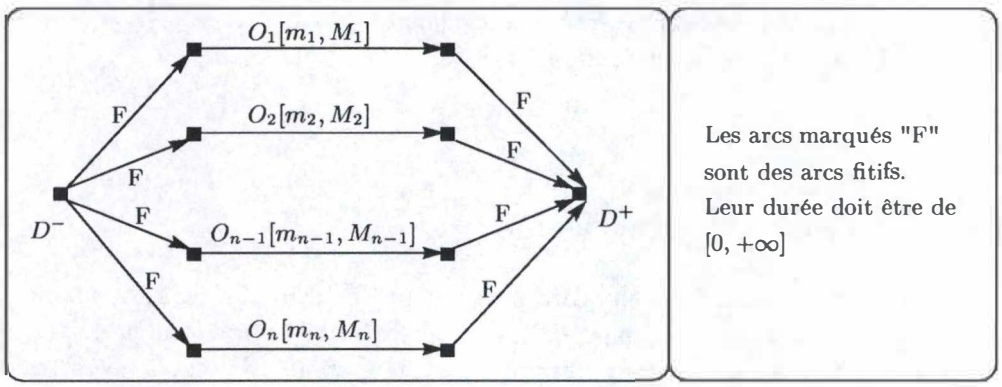


FIG. 7.1 – *Graphe temporel après insertion des médias $O_1, \ldots, O_n$ d'un document D à son chargement, avant insertion de ses relations.*

7.2.2 Insertion des relations

La figure 7.1 nous montre le document après insertion de ses médias. Il reste à y insérer les relations entre éléments du document. Considérons les relations d'intervalles, celles d'Allen, qui ont été étendues pour contenir des paramètres de durées. Chacune d'elle est transformable en plusieurs relations d'instants, selon la figure 3.8 page 52. Pour procéder à l'insertion des relations spécfiées par l'auteur, Madison commence par les convertir en relations d'instants. Il reste à insérer ces relations d'instants dans le graphe.

Insertion de délais et de leur durée

En fait, c'est un rien plus compliqué. Les relations "<" et ">" sont souvent paramétrées par des durées. Elles génèrent des délais. Ces délais doivent être insérés au bon endroit dans le graphe temporel. S'ils n'ont pas de durée les accompagnant, il faut leur en fournir une. Les

arcs de délais ne peuvent contenir la valeur 0. Si tel était le cas, la sémantique des relations d'Allen serait modifiée. En effet, c'est la durée non nulle du délai entre les objets A et B qui différencie une relation "A meets B" d'une relation "A before B". Une fois encore, la direction des arcs de délai est connue. Pour ne pas recevoir de résultats surprenants auprès de PC2, il faut interdire les valeurs négatives pour les durées de ces arcs de délais. Ces arcs doivent donc recevoir une étiquette de durée par défaut de $[1, +\infty]$. La figure 7.2 représente un exemple de document comportant 3 médias A, B et C, plus une relation "B before C" sans argument numérique. Il va donc falloir insérer un délai X entre B et C. Sa durée n'ayant pas été précisée, elle *doit* être de $[1, +\infty]$.

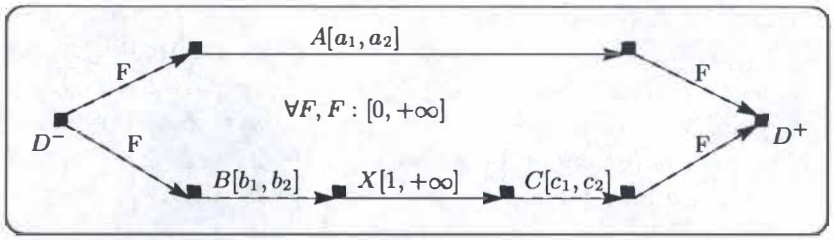


FIG. 7.2 – *Graphe temporel : insertion de délais lors de l'insertion de la relation "B before B". Le délai X inséré* doit *avoir la durée* $[1, +\infty]$.

Fusion de sommets du graphe

Les relations "$a = b$" sont encore plus complexes. En effet, elles signifient la simultanéité d'instants a et b. Cela implique que les sommets représentant a et b doivent être fusionnés. Cette opération n'est pas très courante sur un graphe. Elle est sujette à erreur. En effet, la modification de la topologie du graphe n'est pas suffisante. Bien sûr, il faut que tous les arcs entrants (resp. sortants) du sommet fusionné reprennent tous ceux qui entraient (resp. sortaient) des sommets a et b. Mais il faut de plus être très attentifs aux durées des arcs modifiés. Supposons qu'un arc $X : [x_1, x_2]$ était arc entrant de a et qu'un arc $Y : [y_1, y_2]$ était arc entrant de b. Supposons également que les arcs X et Y ont le même sommet de départ. La figure illustre cette situation.

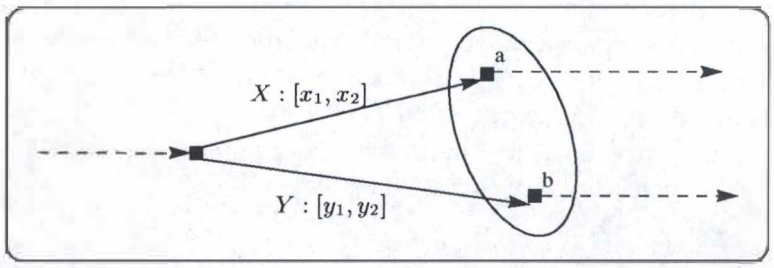


FIG. 7.3 – *Graphe temporel : problème lors de la fusion de deux sommets a et b. Que faire de la durée fusionnée des arcs X et Y ?*

7.2.3 Types de durées à différencier

Voilà comment se passe grosso modo la construction du graphe temporel. Les nombreux détails sortent du cadre de ce mémoire, mais ils en valent la peine. Il faut consulter les 9 classes Java de la partie graphe de Madison pour en savoir plus. Concernant les durées, il faut différencier les cas $[0, +\infty]$ des arcs fictifs, les cas $[1, +\infty]$ des arcs de délais, et les cas $[-\infty, +\infty]$ pouvant se produire dans l'exécution de PC2. La construction du graphe temporel supposait, lors de la création des arcs, une valeur par défaut pour sa durée. C'est cela qui a constitué un bug pendant un bon moment lors de l'implémentation de PC2. En effet, la valeur par défaut est passée de $[0, +\infty]$ à $[-\infty, +\infty]$ parce que PC2 avait besoin de ce type de durées. mais ensuite, les arcs fictifs insérés lors du chargement d'un document utilisaient aussi cette valeur $[-\infty, +\infty]$, alors que leur direction était connue. Cela générait des graphes minimaux pas assez précis : il subsistait des arcs à durée négative après formatage, ce qui "plantait" le formateur temporel.

La description de la construction du graphe temporel valait donc la peine d'être soulignée. Elle est importante pour la correction des algorithmes. En me rappelant du cas spécial de fusion de sommets de la figure 7.3, je suis pratiquement convaincu qu'il susbsiste des bugs dans Madison. Cela doit être le cas lors de l'insertion de certaines configurations de relations temporelles, lors de la modification du graphe temporel.

7.3 PC2 non hiérarchique : problèmes d'implémentation

Cette section aborde des problèmes survenus lors de l'implémentation de la première mouture de PC2, celle qui ne prenait pas encore en compte la hiérarchie. Ceux-ci sont liés à quatre facteurs : la complétude du graphe (7.3.1 p. 89), les arcs multiples entre deux mêmes sommets (7.3.2 p. 90), la taille du logiciel (7.3.3 p. 92) et le langage Java lui-même 7.3.4 p. 93).

7.3.1 Graphe complet

Le premier problème d'implémentation concerne le graphe complet exigé en précondition de PC2 : PC2 exige un graphe acyclique orienté complet. L'orientation des arcs est présente de façon naturelle dans Madison. Le caractère acyclique est assuré par une vérification a priori lors de l'insertion d'une nouvelle relation ou lors de la construction initiale du graphe, au chargement d'un document. C'est la vérification de cohérence qualitative. Par contre, le graphe temporel utilisé par Madison n'est pas complet. Cela serait inimaginable de demander à l'auteur de le faire. En effet, pour un graphe à n nœuds, un graphe orienté complet comporte pas moins de $\frac{n.(n-1)}{2}$ arcs! Il va falloir gérer ce problème.

Le graphe que Madison utilise en interne pour représenter la dimension temporelle et hiérarchique d'un document ne sera jamais complet. En effet, si l'auteur spécifie un média A de durée $[m, M]$ correspondant à un arc orienté i, j dans le graphe, Madison ne conservera pas l'arc i, j opposé. Pourquoi le faire, puisqu'il est aisément calculable : l'arc inverse aura une durée de $[-M, -m]$, comme nous l'avons montré plus haut. De plus, le fait d'insérer cet arc dans le graphe pose un problème supplémentaire. Si la durée de l'arc i, j est modifiée, il faut la répercuter immédiatement sur l'arc inverse j, i. Madison n'a pas besoin de la difficulté supplémentaire de synchronisation entre deux valeurs redondantes. Le graphe temporel de Madison ne sera donc jamais complet.

En conclusion, il faut compléter le graphe. Pour cela, je vois deux sortes de solutions. La première est simple. Elle consiste à ajouter tous les arcs manquant au graphe initial, puis à les enlever à la fin de l'algorithme. Mais cela pose le problème de la synchronisation entre deux valeurs redondantes : celles des durées d'un arc et de l'arc opposé. C'est pourquoi j'ai choisi une deuxième solution. Celle-ci consiste à fournir au vol les bonnes durées. Ceci est réalisé par une méthode Java. Elle accepte en entrée deux sommets i, j et renvoie la durée de l'arc les joignant. Cette méthode utilise le principe suivant :

1. Si l'arc i, j existe, c'est le cas simple. Il suffit de renvoyer la durée que porte cet arc.
2. Si l'arc i, j n'existe pas, mais que l'arc j, i existe avec une durée $[m, M]$, il suffit de renvoyer la durée de l'arc inverse, sans le créer. Cette durée est $[-M, -m]$.
3. Si ni l'arc i, j ni son opposé j, i n'existe dans le graphe, il faut créer un nouvel arc dans le graphe. Il faut veiller à ce que cet arc ne soit pas inséré de manière permanente, car Madison doit récupérer un graphe topologiquement identique, avec comme seule différence les valeurs des durées associées aux arcs. Ces valeurs peuvent uniquement être restreintes, car PC2 utilise une propagation des contraintes par précision de leur domaine (élimination des valeurs de durées n'appartenant à aucune solution). Il suffit de créer un nouvel arc i, j avec un marquage spécial, destiné à le supprimer en fin d'exécution de l'algorithme PC2. Il faut être vigilant : l'arc nouvellement crée *doit* être étiqueté avec une durée $[-\infty, +\infty]$. En effet, la direction de l'arc orienté n'est pas encore connue. Notons que PC2 utilise des valeurs négatives, ce qui n'était pas prévu dans la théorie!.

Nous venons de régler le problème du graphe complet au départ. Mais ce n'est pas le seul problème auquel il faut faire face. La théorie en a éclipsé d'autres.

7.3.2 Arcs multiples entre mêmes sommets

Le second problème d'implémentation, caché lui aussi, est dû à la théorie. Elle n'utilise implicitement que des "1-graphes", càd des graphes pour lesquels il n'existe jamais plus d'un arc entre deux mêmes nœuds. Une fois de plus, ceci n'est pas le cas dans la pratique.

Supposons que l'auteur spécifie une relation de simultanéité entre deux médias, ce qui est très fréquent. Le petit dessin de la figure 7.4 n'est si innocent. La méthode qui vient juste d'être développée pour renvoyer la bonne durée doit être revue. Dans le cas de l'exemple, faut-il renvoyer la durée de A, celle de B, ou une autre encore?

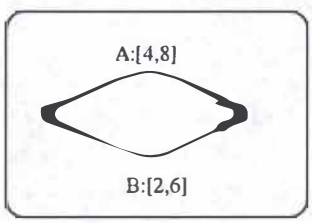


FIG. 7.4 – *Problème de PC2 : les arcs multiples entre deux mêmes nœuds*

Il faut gérer ce problème supplémentaire. Nous sommes dans un contexte de vérification de cohérence. Il se peut que la durée de A ne soit pas compatible avec celle de B, si ce sont des intervalles disjoints. Dans ce cas, la méthode ne doit pas renvoyer de durée, mais arrêter l'exécution du mécanisme de vérification de cohérence. De plus, cette erreur est très simple à

situer et à comprendre pour l'auteur : ce sont deux objets à jouer en même temps, alors qu'ils ont des (intervalles de) durées incompatibles. Il faut donc également renvoyer à l'auteur une description précise de l'endroit où l'erreur s'est produite : entre quels objets (éventuellement plus que deux, d'ailleurs), dans quel composite, etc.

Comment faire au juste ? Il suffit d'adapter légèrement le corps de la méthode que nous venons de donner. Lorsque l'on parle de l'intervalle (unique) entre deux points, il suffit de parler d'un intervalle (quelconque) entre deux points. Lorsque l'on parle de la durée d'un arc, il faut en calculer une autre. La durée à choisir est imposée par la sémantique du graphe.

La bonne durée entre deux sommets

Comment calculer la bonne durée entre deux sommets ? Soient $d_1, \ldots, d_n$ les durées des arcs existant entre deux sommets i et j. La valeur de durée à renvoyer pour "la" valeur de durée entre ces sommets est $d = d_1 \cap d_2 \cap \ldots \cap d_n$. Nous pourrions même examiner le cas où, entre deux sommets i et j, il existerait un ou plusieurs arcs de direction opposée. Mais nous supposerons que ce cas est impossible dans Madison, car il ne se présente qu'entre médias simultanés. Je ne parviens pas à imaginer un scénario où un tel cas pourrait bien se produire. Nous nous contenterons donc de la définition qui vient d'être donnée pour la durée entre sommets liés par plusieurs arcs. Résumons-nous.

Algorithme : renvoi de la durée entre deux sommets

La méthode renvoyant la durée située entre deux sommets i et j est la suivante :

1. Si un arc i, j existe. Calculer la durée d entre les arcs existants s'il en existe plusieurs, sinon d est la durée du seul arc i, j.

   (a) Si l'intervalle de durée d est vide, arrêter PC2, et situer l'erreur à l'auteur : lui donner la liste des médias en conflit (ceux entre i et j).

   (b) Sinon, renvoyer d

2. Si aucun arc i, j n'existe, mais qu'un arc j, i existe. Calculer la durée d entre les arcs existants entre i et j.

   (a) Si d est vide, arrêter PC2 et donner la liste des médias en conflit à l'auteur (ceux entre j et i

   (b) Sinon soit $[m, M] = d$. Renvoyer la durée de l'arc inverse, sans le créer. Cette durée est $[-M, -m]$

3. Si aucun arc i, j ou j, i n'existe dans le graphe.

   (a) Créer temporairement un nouvel arc i, j dans le graphe, pour le supprimer en fin d'exécution de PC2

   (b) Etiqueter i, j avec la durée $[-\infty, +\infty]$.

   (c) Renvoyer cette durée $[-\infty, +\infty]$.

7.3.3 Taille importante de Madison

J'arrive à un troisième type de problèmes d'implémentation : la taille du logiciel. Implémenter une version de PC2, restreinte aux documents sans hiérarchie, crée de purs problèmes d'implémentation : les algorithmes vus doivent être insérés dans le code Java de Madison. Or, ce n'est pas vraiment une petite application.

Exemple : taille de la partie "graphe" de Madison

Je prends l'exemple de la partie "graphe", que je connais bien. Pour implémenter un graphe, Madison utilise trois classes Java : une classe pour les nœuds, une pour les arcs et une pour le graphe.

Ce n'est pas tout. Ces arbres sont synchronisés, pour le support multi-vues de Madison. En effet, un document peut être visualisé en Madison selon plusieurs vues. Une vue "structure" dévoile la hiérarchie du document. Une vue "exécution" sert à son édition directe et à sa présentation. Une vue "scénario" visualise des solutions temporelles du document. Toutes ces vues sont synchronisées : une opération d'édition effectué dans l'une d'elle est directement prise en compte et répercutée sur les autres vues ouvertes simultanément. Ceci est réalisé en synchronisant la structure de données sous-jacente : un graphe de référence. Lorsqu'il est modifié, les répercussions sont apportées à tous les graphes qui se sont "abonnés" à ce graphe de référence. Ce mécanisme de synchronisation par abonnement d'événements est transparent : il est défini dans la classe graphe.

A partir de là, des classes héritées sont définies : il existe plusieurs sortes d'arbres. Concrètement, il existe un arbre générique (synchronisé), représenté par les 3 classes citées. Il existe également un arbre spécifique à Madison, qui hérite du précédent. Il est synchronisé sans avoir à s'en soucier, grâce aux vertus de l'héritage. Cela fait 3 classes en plus. Il existe enfin un arbre spécifique au langage Madeus, car Madison supporte aussi le langage SMIL. Avec ces 3 classes supplémentaires, nous arrivons à 9 classes rien que pour la simple structure de graphe!

Orthogonalité des classes

Le grand nombre de classes permet de factoriser le travail, notamment via des logiciels de gestion de version, comme le merveilleux CVS. Mais il faut beaucoup de temps pour comprendre suffisamment la structure du logiciel. La taille importante d'un logiciel complique sa rédaction.

Voici un premier problème soulevé : les classes ne sont jamais tout à fait orthogonales : il existe des interactions entre elles. Ainsi, PC2 était dépendant du graphe temporel, en particulier de la création des étiquettes de durées de ces arcs. Dans ce cas précis, il s'agissait de créer par défaut des durées de $[-\infty, +\infty]$ dans un cas, et de $[0, +\infty]$ dans un autre. Ce bug n'a pas été trouvé en 5 minutes.

La bonne méthode

Un second problème inhérent à la taille importante de Madison est le suivant. Il faut utiliser chaque fois la bonne méthode de la bonne classe. S'il existe une hiérarchie de classe, il faut choisir le bon niveau d'héritage. Une fois la bonne classe trouvée, il faut savoir exactement quelle méthode utiliser. Faut-il une méthode propre, héritée (plus difficile à trouver si la documentation n'est pas bien gérée), appelée via un objet contenu dans l'instance de la classe?

Bref, il faut du temps pour savoir où trouver les méthodes à appeler, les bons champs à utiliser, etc.

7.3.4 Le langage Java

Voici la quatrième et dernière difficulté rencontrée pour implémenter PC2 : le langage Java. Cette dernière difficulté, même si elle n'est pas la plus grande, nécessite du temps. Il s'agit de l'apprentissage du langage Java, que je ne connaissais absolument pas avant d'arriver à l'INRIA. J'ai mis un certain temps avant de l'assimiler à un niveau suffisant pour me lancer dans l'aventure. Ce langage se veut une simplification du langage C++, avec garde-fous additionnels. Il lui ajoute une notion d'interface, bien pratique. Mais Java cache bien son jeu. A priori pas très compliqué, ce langage comporte tout de même de nombreuses règles d'interprétation, munies évidemment d'exceptions. Par exemple, le test d'une instruction conditionnelle est évalué de plusieurs manières. S'il concerne une constante booléenne, Java suppose qu'il peut s'agir d'une constante de débuggage. Alors, il l'évalue statiquement et ne génère que le code nécessaire. Si ce n'est pas le cas, le test est évalué à l'exécution, ce qui est conforme à la règle générale.

Le langage Java est un langage de programmation orienté objet. Il n'est certes pas aussi déconcertant pour les néophytes que les paradigmes fonctionnels ou logiques en programmation, puisqu'il fait partie du paradigme impératif. Mais les approches orientées objets sont très différentes des autres approches impératives (comme C, Pascal et consorts). Elles disposent d'une série très intéressante de nouveaux concepts. Le style de programmation est radicalement différent. Je peux dire qu'il repose sur une "philosophie" de la programmation radicalement différente. Il faut du temps pour s'y accoutumer.

Les informaticiens de ma promotion et moi-même n'avons pas reçu une formation suffisante concernant Java et les langages orientés objet. C'est dommage. Heureusement, la génération suivante a pu se frotter à Java lors d'un travail pratique. J'espère que l'institut suivra les progrès en informatique afin de les introduire dans les horaires de cours au moment opportun.

Mais tout ceci commence à être hors sujet. Revenons à nos moutons. Je viens de donner des indications concernant l'implémentation de PC2 aux documents plats, non hiérarchiques. Il est temps de passer aux documents hiérarchiques.

7.4 Extension à la hiérarchie

PC2 a été présenté pour la détection des incohérences et pour le calcul du graphe minimal dans le cas où la cohérence est vérifiée. Mais il l'a été dans le contexte restreint des documents sans hiérarchie. Ce nouvel aspect, la hiérarchie des documents, a été étudié par l'un de mes prédécesseurs en stage à l'INRIA, Frédéric Bes. Il a d'ailleurs rédigé un rapport à ce sujet [Bes98].Ce sont ses idées qui sont à la base de cette section et du travail que j'ai effectué.

7.4.1 Graphe temporel hiérarchique

Le problème de la gestion temporelle de documents hiérarchiques a été résolu par l'adoption d'un graphe temporel hiérarchique.

La hiérarchie du document est introduite via la notion d'éléments composites. Par abus de langage, j'appelle *arc basique* un arc représentant un élément basique (un média ou un délai).

De la même manière, j'appelle *arc composite* un arc représentant un élément composite (ceux qui en contiennent d'autres, générant récursivement un arbre hiérarchique).

La structure décrite précédemment pour PC2 employait un seul graphe, le graphe global. Cette même structure de données va être réutilisée pour l'extension à la hiérarchie, mais uniquement pour représenter des graphes locaux. En effet, à chaque niveau de hiérarchie sera associé un graphe local. Plus précisément, c'est chaque élément composite qui possèdera un graphe local le représentant. Le document dans son totalité sera représenté par une sorte d'hypergraphe. Il s'agit d'un graphe global composé d'arcs basiques et d'arcs composites. Ces arcs composites possèdent chacun une référence vers un graphe local représentant le composite décrit sur cet arc. A leur tour, chacun de ses graphes locaux peut récursivement comporter des arcs composites renvoyant à d'autres graphes locaux. Un document multimédia structuré représenté dans Madison peut être vu dans sa dimension temporelle hiérarchique comme un arbre classique dont les nœuds sont des graphes, tout comme les feuilles.

Un exemple

Tout cela a l'air bien compliqué, mais il n'en est rien. Un petit exemple va nous éclairer. Considérons le document hiérarchique de la figure 7.5. Sa hiérarchie est simple : le document D se compose de trois scènes S_1, S_2, S_3. Chacune de ces scènes se compose de trois objets. Le document comporte des relations temporelles simples. Chaque relation entre deux éléments est spécifiée au niveau du composite père, celui dont les éléments en relations sont les fils direct. Rappelons-nous que Madeus n'autorise pas l'insertion de relation autres que celles entre frères de même niveau dans l'arbre hiérarchique. Il faut donc exclure toute relation entre éléments de niveaux différents, même s'ils ont un père commun.

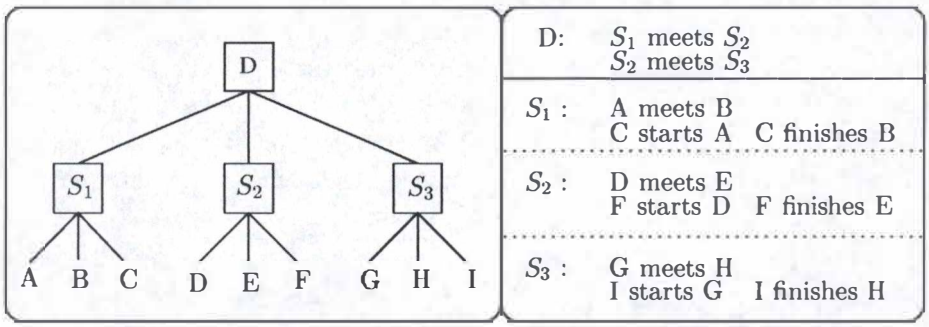


FIG. 7.5 – *Graphe hiérarchique. Exemple d'un document hiérarchique avec ses relations temporelles*

Voici les relations de l'exemple. La scène S_1 se compose des éléments A, B et C liés par une relation "meets" entre A et B, une relation "start" entre A et C, ainsi qu'une relation "finishes" entre C et B. Autrement dit, les médias A et B se suivent, pendant qu'un autre média C est présenté. Des relations symétriques sont définies pour les composites S_2 et S_3. Enfin, au niveau du document entier, l'élément composite D contient les relations entre ses éléments, les 3 scènes A, B et C. Les éléments A et B sont présentés simultanément (A equals B). Ensuite vient l'élément C, sans délai après la fin de A et B (relation A meets C). Ces relations sont listées également sur la figure 7.5.

Les objets composites sont les scènes S_1, S_2, S_3 et le document D lui-même. Associons un graphe temporel à chacun d'eux. Le document peut être vu comme un arbre hiérarchique dont

les nœuds sont les graphes temporels des composites. C'est ce qu'illustre la figure 7.6. Même avec ce tout petit exemple, nous voyons que le fait de disposer d'un mécanisme d'abstraction (hiérarchie) constitue un atout indispensable. Sans cela, on se noie extrêmement vite dans les détails. La figure 7.7 montre le document tel qu'il serait si la représentation de la dimension temporelle n'était pas effectuée hiérarchiquement par un hypergraphe : il devient vite difficile de s'y retrouver.

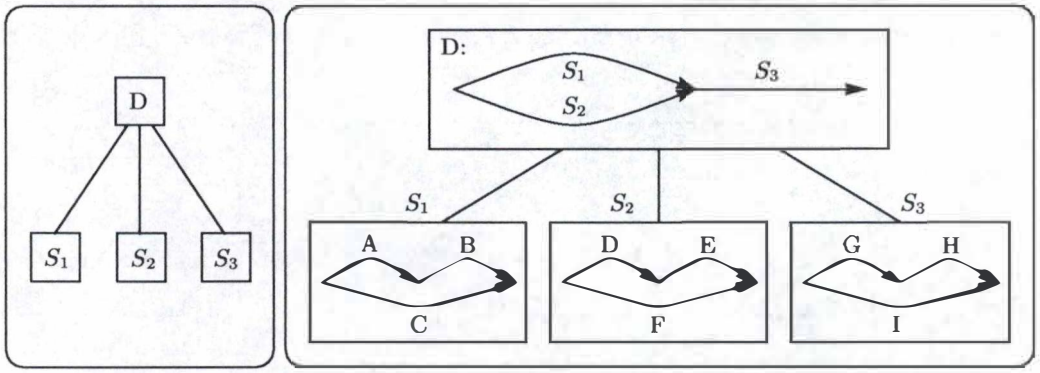


FIG. 7.6 – *Graphe hiérarchique : hypergraphe représentant le document hiérarchique précédent*

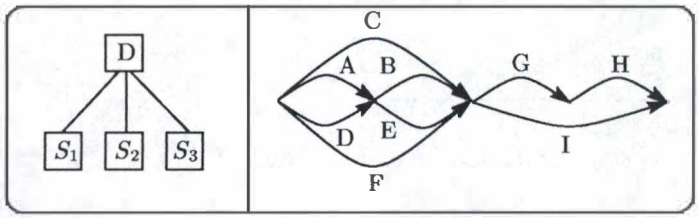


FIG. 7.7 – *Graphe hiérarchique : représentation du document hiérarchique sans hypergraphe (avec un seul graphe)*

Structure de données

Maintenant que les idées sont fixées, je peux passer à la structure de données réellement employée dans Madison. Celle-ci servira à comprendre l'extention de PC2 aux documents hiérarchiques. La seule nuance supplémentaire à ajouter pour obtenir la représentation interne du graphe temporel est le lien entre l'arc composite et le graphe représentant ce composite dans le graphe de niveau inférieur. Tout composite dispose d'un graphe le représentant. Mais ce composite doit avoir un instant de début et un instant de fin. De fait, le fait de représenter un graphe de niveau inférieur, contenu via un arc dans le graphe supérieur, de la même manière que les autres arcs permet de leur apporter des traitements similaires. Devant la nécessité de désigner un début et une fin aux composites, Madison ajoute un arc fictif à tout composite. Cet arc fictif relie deux sommets : les sommets de début et de fin du composite.

Il se pose alors un problème de choix pour l'implémentation. Considérons la figure 7.8. Nous y voyons la véritable représentation en Madison d'un graphe temporel hiérarchique. Il existe deux arcs pour représenter un composite. Le premier est l'arc fictif représentant le

composite dans le graphe qui figure son contenu ("a" sur le schéma). Le second est l'arc représentant le composite comme élément dans un graphe qui le contient ("A" sur le schéma). Pour l'implémentation, une question importante se pose. Faut-il que ces deux arcs soient les mêmes au sens objet (même emplacement en mémoire)? Cela fait une différence importante dans l'implémentation de PC2, si l'on n'y prend garde: cela ferait un joli bug bien difficile à débusquer si l'on ne connaît pas avec précision la représentation exacte du graphe temporel. Dans mon implémentation, j'ai essayé de les considérer comme s'ils étaient différents. Ainsi, dans chacun des deux choix possibles, je devrais obtenir des résultats cohérents. Le seul inconvénient est la présence d'affectations qui sont inutiles dans le cas où les deux arcs sont déjà égaux.

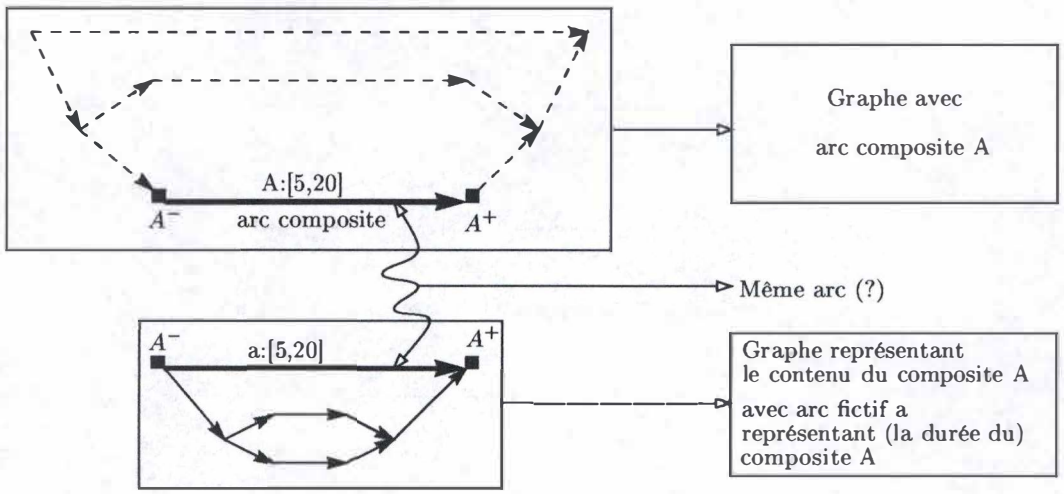


FIG. 7.8 – *Graphe hiérarchique : véritable représentation d'un graphe temporel hiérarchique dans Madison*

Durée d'un composite

Il existe un autre aspect auquel on ne pense pas forcément. Il a pourtant une grande importance sur la sémantique d'un document. Il s'agit d'attribuer un intervalle de validité à l'arc fictif représentant le composite. Quelle la valeur de durée faut-il lui donner? La composition des durées sur le chemin partant du sommet début au sommet fin paraît une bonne réponse. Mais il y a plusieurs chemins. Nous pouvons choisir une durée $[m, M]$ où m et M sont respectivement les minimum et maximum trouvés sur tous les chemins. Il se peut alors que m et M soient des valeurs prises sur deux chemins différents. [1] Nous pouvons également choisir d'anticiper sur la vérification de cohérence et choisir comme durée l'intersection des durées sur tous les chemins menant du début à la fin du composite. Enfin, nous pouvons laisser la valeur telle qu'elle a été choisie par l'auteur, esquivant ainsi tous les problèmes. C'est ce qui est fait dans Madison pour l'instant.

Il y a encore un autre problème de définition précise: quels sont au juste les instants de début et de fin d'un composite? Il existe deux choix pour le faire. Le premier est de dire que le

1. Considérons par exemple un composite où figurent deux chemins distincts de (somme de) durée [4,10] et [8,20]. Faut-il choisir [4,20] comme durée (sur des chemins différents)? Ou bien anticiper la vérification de cohérence et affecter l'intersection [8,10]?

début du composite doit commencer avant celui de ses éléments constitutifs et que sa fin doit suivre celle de ses éléments. Cela peut conduire à des résultats surprenants. Si par exemple l'auteur demande une relation "A equals B" où A est un média et B un composite. Dans ce cas, il est possible que la présentation commence par le média A, attende un moment, puis présente le premier élément du composite B. Cela respecte cette première sémantique, définie comme suit. Soient $C_1, \ldots, C_n$ les éléments constitutifs d'un composite C. Alors, il faut vérifier les inégalités suivantes :

$$\begin{aligned} C^- &\leq \min(C_i^-) \\ \max(C_i^+) &\leq C^+ \end{aligned}$$

Le second choix est plus restrictif. Intuitivement, il fait coïncider le début d'un composite avec le début du composant commançant le plus tôt et sa fin avec celle du composant terminant le plus tard. Il faut vérifier cette fois les égalités :

$$\begin{aligned} C^- &= \min(C_i^-) \\ \max(C_i^+) &= C^+ \end{aligned}$$

D'où proviennent donc ces imprécisions? Le problème, c'est qu'il faut pouvoir définir le début (et la fin) d'un composite. Or, si le début d'un composite est celui du premier de ses éléments qui commencent, il faut connaître à l'avance lequel des éléments sera le premier. Cela n'est pas toujours possible statiquement. La seule chose certaines, c'est que :

$$C^- = C_1 \vee C^- = C_2 \vee \ldots \vee C^- = C_n$$

Diantre, voici de nouveau les fameuses disjonctions d'instants! Cela n'est pas simple. Je me contenterai d'ignorer le problème pour l'instant, comme c'est fait provisoirement dans Madison.

7.4.2 Algorithme PC2 hiérarchique

Avec cette nouvelle structure de données, PC2 est prêt à être étendu aux documents hiérarchiques. La difficulté consiste à propager les contraintes en tenant compte cette fois de la hiérarchie. L'idée de base consiste à réutiliser le PC2 non hiérarchique sur les graphes locaux. Ensuite, il faut assurer les propagations entre niveaux. Cette propagation peut se dérouler de deux manières : de bas en haut (bottom-up) ou de haut en bas (top-down), en faisant allusion au sens de parcours de l'arbre hiérarchique. Dans la suite, je définis deux types de cohérence hiérarchique, et des propriétés qu'elles respectent. Je continue en décrivant les diverses formes d'algorithme vérifiant la cohérence hiérarchique.

Types de cohérence hiérarchique

Pour vérifier la cohérence, j'ai choisi et implémenté deux types d'algorithmes de vérification de cohérence "hiérarchique" dans Madison. J'ai baptisé ces deux versions selon le type de cohérence hiérarchique qu'ils assurent. Je les ai appelées :

- *Cohérence forte*. C'est la cohérence telle que précédemment définie, étendue aux documents hiérarchiques. Les domaines sont toujours minimaux localement (pour chaque

composite). Mais ils sont encore minimaux globalement. Autrement dit, chaque arc composite de tout graphe a la même valeur que l'arc représentant ce composite dans le graphe de niveau inférieur, et tous ces graphes sont minimaux.

- *Cohérence faible*. C'est une forme restreinte de cohérence. Les domaines sont minimaux localement, au niveau de chaque composite. Mais ils ne le sont plus globalement. Autrement dit, l'arc composite d'un graphe peut avoir une durée différente que celle de l'arc le représentant dans le graphe inférieur. Cela permet de détecter les incohérences plus rapidement, sans calculer les graphes minimaux globalement.

Illustration à l'aide d'un exemple

La différence entre les deux n'est pas évidente. Je vais donc l'illustrer sur un exemple, sur le schéma 7.9. La méthode employée est ascendante (bottom-up). En 'a', c'est le graphe temporel initial. Le dessus de la figure représente un arc composite X, tandis que le dessous représente le graphe de ce composite. Ce dernier comporte l'arc x représentant la durée du composite X du graphe supérieur. L'algorithme considère l'arc composite X. En bottom-up, il faut d'abord lancer PC2 sur le graphe inférieur. (En top-down, c'est le contraire.) PC2 est donc lancé sur le graphe inférieur, générant le graphe résultat en 'b'. La propagation consiste à donner à l'arc X la valeur (plus précise) qui a été trouvée pour l'arc x du graphe inférieur. C'est l'étape 'c'. Enfin, PC2 continue sur le graphe supérieur pour préciser encore l'arc X, sans plus se soucier du graphe inférieur. Ceci est la cohérence hiérarchique faible.

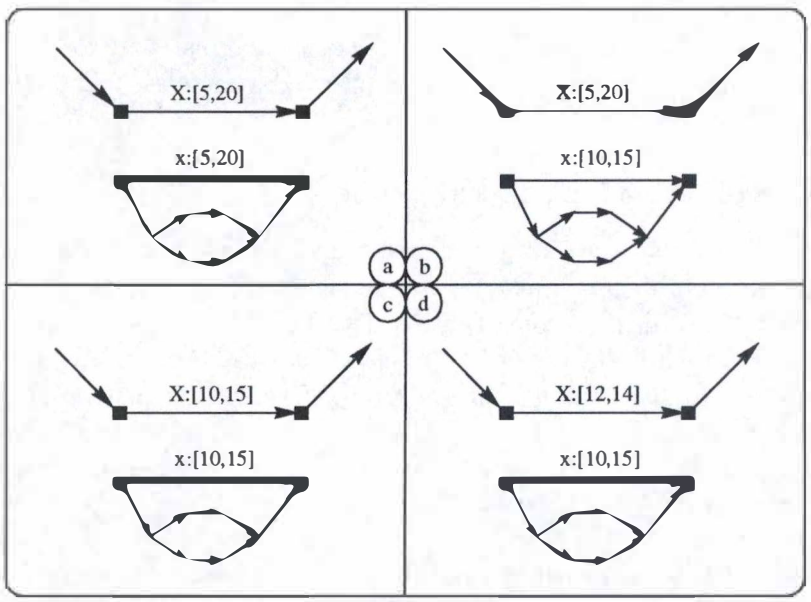


FIG. 7.9 – *Cohérence hiérarchique faible : exemple bottom-up en 3 étapes. Le graphe 'b' est obtenu après exécution de PC2 sur le le graphe inférieur 'a'. La propagation bottom-up génère le graphe 'c'. Le graphe "faiblement" cohérent est obtenu en 'd' après réexécution de PC2 sur le graphe supérieur.*

Propriétés des cohérences hiérarchiques

Pour mieux comprendre encore ces deux types de cohérence, je donne ici leurs propriétés.

La cohérence faible est suffisante pour détecter des incohérences entre niveaux hiérarchiques, mais pas pour obtenir des graphes minimaux valables localement. Reprenons l'exemple précédent, sur la figure 7.9. Le graphe supérieur est cohérent pour l'arc X s'il a une durée de [12, 14]. Le graphe inférieur est cohérent pour l'arc principal x s'il a une durée de [10, 15]. Cela ne pose pas de problème de cohérence, car leur intersection est [10, 14], non vide. De plus, si le composite est cohérent dans l'intervalle [10, 15], il le sera forcément encore dans un intervalle plus restreint, comme [10, 14]. La seule condition véritable est l'intersection non vide entre les valeurs des arcs x et X.

Il existe une autre façon d'expliquer la cohérence faible. La propagation des contraintes de PC2 est une technique de filtrage : elle élimine des intervalles les valeurs ne pouvant appartenir à aucune solution.

La cohérence hiérarchique faible filtre les graphes locaux sans étendre le filtrage à la hiérarchie. Il existe des solutions dans certains graphes qui ne le sont pas dans d'autres. Dans l'exemple précédent (figure 7.9), le choix de la durée [11, 11] est valable dans le graphe contenant l'arc x, mais pas dans celui contenant l'arc X.

La *cohérence hiérarchique forte* est une cohérence faible avec un petit ajout. La propriété supplémentaire consiste à filtrer le graphe globalement en éliminant les valeurs qui ne sont pas solutions dans tous les graphes. Dans le cas de l'exemple précédent (toujours sur la figure 7.9), il faut que les arcs x et X aient la même durée. Il faudrait donc relancer PC2 sur le graphe inférieur, après avoir affecté la durée [12, 14] à l'arc x.

Il existe une propriété intéressante pour ces cohérences hiérarchiques. Avec une méthode ascendante (bottom-up), comme celle de l'exemple de la figure précédente, la durée de l'arc X du graphe supérieur sera toujours plus précise (petite) que celle de l'arc x du graphe inférieur. Avec une méthode descendante (top down), c'est le contraire.

Comment cela fonctionne-t-il?

Voici enfin le principe des algorithmes de vérification de cohérence hiérarchique.

La *cohérence faible* est simple et efficace. Il s'agit de lancer PC2 selon une approche bottom-up. C'est le même algorithme que PC2 non hiérarchique, aux différences suivantes près. Quand il rencontre un arc composite, il faut d'abord lancer PC2 sur le graphe inférieur correspondant à ce composite. Ensuite, il faut récupérer la nouvelle valeur de durée de l'arc composite du graphe inférieur pour l'affecter à l'arc composite rencontré dans le graphe supérieur. (bottom-up). Ceci dit, c'est un rien plus compliqué qu'une affectation. En réalité, il faut prendre l'intersection de ces deux valeurs. Si cette intersection est vide, cela signifie qu'une incohérence de type purement hiérarchique est intervenue. Sinon, l'algorithme continue là où il en était dans le graphe supérieur, en considérant la nouvelle valeur obtenue comme valeur de durée de l'arc composite rencontré.

Si l'approche était top-down, cela serait légèrement différent. Lorsqu'un arc composite est atteint, l'algorithme PC2 continue comme dans le cas non hiérarchique. Ensuite, lorsqu'il en a terminé avec un graphe, il considère tous ses arcs composites et leurs graphes associés. Pour chacun de ces graphes, le procédé recommence comme il vient d'être décrit. Il s'agit toujours d'une cohérence faible. Simplement, la durée obtenue pour un composite sera plus précise dans le graphe inférieur que dans le graphe supérieur.

La *cohérence forte* est plus compliquée à gérer. Elle est pourtant indispensable pour formater un document hiérarchique, car il faut disposer des graphes minimaux globalement pour ne pas remettre en cause une instanciation lors de la construction d'une solution. Pour mettre les graphes d'accord entre eux, il faut intervenir au niveau des arcs composites. Ils sont présents en doublons : celui du graphe supérieur (X) et celui du graphe inférieur (x). Lorsque X est plus précis (bottom-up), il faut affecter sa valeur à x et relancer PC2 sur le graphe inférieur. Lorsque x est plus précis (top down), il faut affecter sa valeur à X et relancer PC2 sur le graphe supérieur. Les détails d'implémentations sont similaires à ceux de la cohérence faible. Il ne faut pas par exemple oublier de prendre l'intersection des valeurs x et X, par exemple.

Comme l'on peut le voir, PC2 hiérarchique s'amuse à lancer des PC2 non hiérarchiques sur des graphes, en se baladant tantôt vers le haut, tantôt vers le bas. Sachant que PC2 non hiérarchique est cubique $O(n^3)$, le temps d'exécution de PC2 hiérarchique a de quoi faire peur.

7.5 Formatage

Comme souligné plus haut, le formatage réutilise l'algorithme PC2. Le formatage est l'opération qui consiste à trouver une solution aux contraintes (spatio-)temporelles d'un document, dans le but d'en effectuer une présentation à l'écran. Les critères de performances sont critiques dans Madison, car le but est de s'approcher du principe du Wysiwig (What You See Is What You Get). Les phases d'édition et de présentation étant obligatoirement dissociées par la nature temporelle du document, il faut disposer d'un temps le plus court possible entre édition et présentation. Le temps de passage ne devrait être dans le pire des cas de l'ordre de la seconde.

7.5.1 Précondition du formatage : une cohérence forte

Pour obtenir une performance suffisante, il est indispensable de ne pas devoir remettre en cause le choix d'une durée lorsqu'il faut en déterminer une autre, ultérieurement. En d'autres termes, il faut disposer des graphes minimaux *globalement*. Cela ne change pas grand chose pour un document non hiérarchique. Mais Madison ne s'en contente pas, car une de ses raisons d'être est la gestion de documents complexes et importants, pour lesquels une capacité d'abstraction (hiérarchie) est primordiale (visualisation facilitée, réutilisation de parties de documents, etc). Bref, la précondition sine qua non du formatage temporel est de disposer des graphes minimaux globalement. La vérification de cohérence peut être faite rapidement à l'aide de la cohérence hiérarchique faible, mais le formatage requiert l'utilisation de la cohérence hiérarchique forte.

7.5.2 Version non hiérarchique

La version non hiérarchique du formatage est assez simple. La précondition est toujours de disposer des graphes minimaux. C'est chose faite avec PC2 non hiérarchique, qui assure la 2-cohérence et les graphes minimaux. Une fois que ceux-ci sont calculés, le choix d'une durée ne sera plus remise en cause ultérieurement, afin d'assurer un formatage dans des temps les plus faibles possibles.

Je me dois de rappeler un détail extrêmement important. La non remise en question ne vaut pas dire choisir simplement une durée pour chaque arc, après avoir calculé les domaines

minimaux une fois pour toutes. Cela conduirait à des incohérences. Il faut recalculer les domaines minimaux après chaque choix d'une valeur de durée. Pour s'en convaincre, il suffit de reconsulter la figure 5.3 page 74 et les figures suivantes, dans le chapitre 5.4 page 73 consacré au lien entre cohérence et formatage.

L'algorithme de formatage non hiérarchique

Le formatage non hiérarchique est donc un algorithme très simple. Après avoir lancé PC2 non hiérarchique pour calculer les domaines minimaux, il choisit une durée pour un arc du graphe. Ensuite, il relance PC2 pour tenir compte de la modification, en initialisant la pile de PC2 avec cet arc. Sachant que ce choix ne sera pas remis en cause, il choisit de fixer la durée d'un autre arc. Puis il initialise la pile de PC2 avec cet arc et relance PC2 non hiérarchique pour tenir compte de cette nouvelle modification, et ainsi de suite.

Performance

Les plus attentifs auront remarqué que ceci signifie grosso modo de lancer PC2 non hiérarchique autant de fois qu'il n'y a d'arc dans le graphe de départ. Il faut donc être très attentifs aux performances de PC2. Cependant, un petit détail oublié vient à notre rescousse. PC2 est relancé sur un graphe où un seul arc a été modifié. Or, PC2 exploite la localité, en arrêtant la propagation le plus tôt possible. En effet, la propagation est stoppée si une nouvelle valeur de durée calculée est identique à la précédente: cfr. les deux tests de différence dans l'algorithme de PC2 (section 6.4.1 page 81). Cela limite déjà un peu le temps d'exécution des PC2 non hiérarchiques.

7.5.3 Version hiérarchique

La version hiérarchique utilise exactement le même procédé. La première chose à faire est de lancer PC2 hiérarchique assurant la cohérence forte. Nous n'avons pas le choix, car il faut disposer des graphes minimaux globaux. C'est la précondition de l'algorithme de formatage hiérarchique. La suite est identique à la version non hiérarchique, ou presque.

L'algorithme choisit la durée d'un arc. Il recalcule les graphes minimaux globaux en lançant PC2 hiérarchique, qui se balade de bas en haut et de haut en bas dans l'arbre hiérarchique. Ensuite, il choisit un autre arc, relance PC2 hiérarchique, et ainsi de suite. La seule différence par rapport à la version non hiérarchique, c'est que le choix des arcs dont on fixe la durée est effectuée au niveau de tous les graphes. Il existe encore plusieurs versions pour le formatage.

Versions de l'algorithme de formatage hiérarchique

Il existe une version ascendante (bottom-up), celle qui commence par fixer les durées des arcs appartenant aux graphes les plus inférieurs. Pour ce faire, il suffit, lors de la rencontre d'un composite, de fixer d'abord les durées des arcs appartenant au graphe inférieur représentant ce composite, pour ensuite revenir à l'arc composite et continuer comme avant.

Il existe également une version descendante (top down). Celle-ci commence par fixer les durées des arcs appartenant aux graphes les plus hauts, avant de s'attaquer aux graphes de niveaux inférieur. Pour cela, l'algorithme fixe les durées de tous les arcs du graphe local. Ensuite, pour chaque arc composite de ce graphe local, il traite chaque graphe inférieur représentant chacun des arcs composites, en fixant les durées des arcs de la même manière, et ainsi de suite.

Les méthodes précédentes fixent un ordre pour le choix des durées des arcs. La version ascendante fixe d'abord les arcs des composites les plus bas avant de remonter. La version descendante fixe d'abord les arcs des composites les plus haut avant de descendre. Il peut exister d'autres sortes d'ordre pour le choix des durées, en se combinant éventuellement avec les deux versions précédentes. Ainsi, il est possible de définir presque n'importe quelle politique de choix de durée. Par exemple, nous pouvons choisir de fixer d'abord la durée des éléments les plus sensibles à la présentation, comme la vidéo et l'audio, avant de continuer avec les médias puis avec les intervalles de délai. Cela n'est qu'un choix parmi beaucoup d'autres. Il ne devrait pas être trop compliqué de proposer plusieurs types de formatage à l'utilisateur. Il suffit de définir une méthode Java qui choisit au moment opportun une des méthodes implémentant la politique de choix. Cette dernière méthode sera appelée à chaque fois qu'il faut choisir la durée d'un nouvel intervalle. Il me semble faisable d'autoriser cette souplesse certaine. Ceci serait une extension intéressante à réaliser, d'autant plus que les auteurs ne comprennent pas forcément la manière dont les solutions sont construites[2].

Performance

Cette fois, il y a plus ou moins autant d'appels à PC2 hiérarchique que de nombre d'arcs dans le graphe global (l'hypergraphe). Sachant que le temps d'exécution de PC2 non hiérarchique est déjà en $O(n^3)$, et que PC2 hiérarchique possède lui-même un temps d'exécution à faire peur, nous voilà devant un problème d'envergure. Bien sûr, la localité est toujours exploitée pour réduire les temps d'exécution. Mais un seul formatage est une opération vraiment très lourde, alors que les exigences de rapidité y sont les plus fortes lors de l'édition d'un document. Les temps d'exécution de l'implémentation actuelle sont encore beaucoup trop élevés. Il faudra trouver des améliorations, car c'est extrêmement important.

7.6 Performances

Je viens de souligner la terrible importance des temps d'exécution des algorithmes précédents. Le formatage hiérarchique doit être rapide alors qu'il fait de très nombreux appels à PC2 hiérarchique, qui lui-même fait de nombreux appels à PC2 non hiérarchique, qui est lui même un algorithme pas très rapide en $O(n^3)$. Atteindre de bonnes performances est donc une tâche aussi capitale que difficile à mener à bien. L'implémentation que j'ai réalisée ne me semble pas encore assez bonne de ce côté-là.

7.6.1 Suggestions pour améliorer les performances

Il faudra apporter des extensions à ces algorithmes. Je pense par exemple à une politique pour l'ordre du choix lors d'un formatage. Il faudra également réaliser des améliorations futures, que je n'aurai pas l'occasion d'effectuer personnellement. Cependant, de par mon expérience à l'INRIA, je peux donner quelques voies vers une exécution plus rapide des algorithmes. C'est par là que je termine ce chapitre: quelques suggestions pour améliorer les performances en temps d'exécution.

2. Heureusement que ces détails leur sont épargnés, d'ailleurs. Sinon, ils auraient tôt fait de s'orienter vers d'autres approches d'édition multimédia.

Accès aux arcs en matrice d'incidence

Un "profiler" Java a révélé que PC2 non hiérarchique passait le plus clair de son temps à accéder aux arcs du graphe. Le graphe est implémenté actuellement sous forme de liste d'arcs et de sommets : chaque sommet possède la liste des arcs entrants et sortants. PC2 a besoin d'accéder aux arcs via les deux sommets a et b à leur extrémité. Pour l'instant, il faut parcourir la liste des arcs entrants et sortants de a pour y trouver éventuellement le sommet b. Si l'on désigne par L la longueur moyenne d'une de ces listes, il faut en moyenne en parcourir la moitié de la liste pour trouver le bon sommet. L'accès à un arc via ses deux sommets se fait donc en $O(\frac{L}{2})$. Dans un graphe complet, chaque sommet est relié aux autres : la liste des arcs accessibles depuis un sommet les comporte tous. Cette liste comporte donc $n-1$ éléments : $L = n-1$.

Pour étudier l'influence d'un arc sur les autres, il faut accéder à tous les sommets k différents de sommets i et j de l'arc dont on étudie l'influence. Si n représente le nombre de sommets dans le graphe, il faut accéder à $2(n-2)$ arcs, car il existe $n-2$ sommets k.

En un mot, le calcul de l'influence d'un arc sur les autres utilise un temps d'accès aux arcs qui est de $2(n-2)(n-1)$. De façon plus concise, c'est un temps en $O(n^2)$.

La solution serait un accès global en $O(n)$, via un accès en $O(1)$ aux arcs. Pour cela, il suffit de coder le graphe en utilisant une matrice d'incidence, càd une matrice dont les lignes et les colonnes sont des sommets, et dont l'élément i, j désigne les arcs d'origine i et d'extrémité j.

Minimiser les allocations dynamiques

Les allocations dynamiques sont très coûteuses en temps. J'en ai fait les frais. La méthode qui calcule la réduction de d'un intervalle par deux autres était capable d'afficher un message d'erreur comportant les identifiants des sommets incriminés en cas de circonstances anormales. Ce message était créé par une allocation dynamiquement ("new" en Java). Par une erreur d'inattention, ce message était calculé (pas affiché, bien sûr) même quand les conditions étaient normales. Cette méthode ("Duration.reduce") est une des plus fréquemment appelées : le nombre d'appels suit le temps d'exécution de PC2 non hiérarchique $O(n^3)$. Lorsque je me suis aperçu de l'erreur, le temps d'exécution a été considérablement amélioré. La plupart des formatages allaient deux à trois fois plus vite !

Il y a sans doute un problème potentiel pour l'accès aux durées des arcs. A cause du caractère complet du graphe, il faut accéder aux arcs en devant considérer à la fois leur deux directions possibles, pour trouver laquelle des deux comporte physiquement un arc. (La durée de l'arc opposé est calculable et pas insérée physiquement dans le graphe.) La durée accédée peut être inversée. Lorsqu'il faut la réinverser, une nouvelle durée $[-M, -m]$ est calculée à partir de la durée $[m, M]$ renvoyée. A ce moment-là, il vaut mieux ne pas créer une nouvelle durée, mais utiliser une référence à la durée précédente. Sinon, le nombre de durées créés dynamiquement va occuper énormément et inutilement le CPU.

Programmation dynamique

Lors du formatage d'un document, il est fréquent de lancer PC2 sur les mêmes parties du graphe, à des moments différents. Il est sans doute possible d'éviter un certain nombre de recalculs en mémorisant les instances de PC2 lancées plutôt que de les recréer, avec des réinitialisations inutiles.

Vérification de cohérence plus rapide

Comme mentionné plus haut, il est possible d'effectuer une vérification de cohérence rapide en préférent une cohérence hiérarchique faible plutôt que forte. L'inconvénient est de ne pas disposer des graphes minimaux. Mais la possibilité est intéressante. Il existe aussi d'autres algorithmes permettant de le faire. L'algorithme de DPC-Cohérence évoqué en page 72 dans la section 5.3.3 du chapitre consacré à la cohérence en fait partie: il est $O(n^2)$ plutôt que $O(n^3)$ comme PC2.

Désactiver les traces de "debug"

La mise au point des logiciels est facilitée par les "débugueurs" et des messages à l'écran décrivant leur états internes. L'affichage de ces messages est lent. Le simple fait de les supprimer accélère la vitesse d'exécution. Lorsque ces traces étaient très nombreuses, le temps d'exécution était réduit d'un facteur allant jusqu'à 5 lors de leur désactivation!

Travailler en opérations désynchronisées

J'ai déjà effectué ce choix lors de mon implémentation. Comme précisé auparavant, le graphe représentant le document est dupliqué et synchronisé avec les vues qui le représentent, pour répercuter les modifications d'une vue sur les autres. C'est géré par un mécanisme d'abonnement à des événements, de manière transparente, grâce à la bonne structure orientée objet de Madison. Mais lors de la modification de la durée d'un arc, il ne faut surtout pas rester synchronisé. Cela ne sert à rien, car les modifications sont excessivement nombreuses. Il suffit de débrancher le mécanisme et de répercuter les modifications de durée d'intervalle une seule fois, à la fin, lorsqu'il n'y en aura plus d'autre. A ce moment, il faut rebrancher la synchronisation des graphes. Le temps gagné est loin d'être négligeable.

7.6.2 Suggestions d'extensions

Voici enfin des suggestions d'extensions à apporter. Les premières extensions indispensables sont celles qui amélirorent les performances en temps d'exécution. Pour les autres extensions, j'ai déjà mentionné plus haut qu'il devrait être fourni à l'auteur le moyen de choisir sa politique pour l'ordre du choix lors d'un formatage. Madison pourrait présenter parmi des options avancées une liste des politiques supportées: ordre aléatoire ascendant, ordre aléatoire descendant, ordre commençant par les vidéos en ascendant, etc. Il y a bien sûr d'autres extensions à réaliser. Ces extensions sont déjà prévues dans Madison: ce sont la gestion des relations causales et de l'indéterminisme. Mais ces problèmes sont complexes, et peu de recherches les considèrent pour l'instant.

C'est pourquoi je vais aborder ce problème dans le chapitre suivant. Mais cela ne sera pas une étude approfondie, car elle sort très largement du cadre de ce mémoire. Tout au plus tenterais-je de résoudre certains cas bien précis.

Chapitre 8

Incontrôlabilité

Nous allons enfin aborder les difficultés et limites qui se présentent lors de la vérification de cohérence. Nous avons traité le cas simple où tout est connu statiquement à l'avance, ce qui écartait les intervalles incontrôlables que nous devons gérer à présent. Ceux-ci soulèvent des problèmes pour lesquels des recherches actives sont en cours, mais non encore abouties parfaitement. De nouveaux concepts ont fait leur émergence, mais le cadre théorique doit encore être développé.

8.1 Définition

Nous nous intéressons aux intervalles incontrôlables. Il est sage de rappeler les définitions associées (déjà données dans la section 5.2.4, page 70 du chapitre consacré à la cohérence). Nous les reformulons et les précisons quelque peu.

- Un *intervalle contrôlable* A : $[m, M]^c$ (représentant un objet A) est un intervalle de validité $[m, M]^c$sur lequel le système d'édition a tout contrôle. Pour un élément A de type discret, il peut choisir arbitrairement une durée dans $[m, M]$. Pour un élément A de type continu, il peut choisir une valeur de durée dans $[m, M]$ à condition qu'elle respecte l'intelligibilité du contenu. Par exemple, un son devient incompréhensible s'il est joué beaucoup trop lentement. Dans ce cas, le système d'édition devrait choisir lui-même les bornes m et M en fonction d'une légère variation autour de la durée "normale" de l'élément, et ne pas permettre à l'auteur de les spécifier.
- La *flexibilité* (càd la différence $M - m$) caractérise la souplesse que l'auteur attribue au scénario.
- Un *intervalle incontrôlable* B : $[m, M]^i$ (représentant un objet B) est un intervalle dont l'intervalle de validité $[m, M]^i$ n'est connu précisément qu'au cours de l'exécution du document. Le système de contraintes revêt un aspect dynamique.
- La *contrôlabilité* est la capacité du scénario à être adapté dynamiquement lors de son exécution. La compensation de l'indéterminisme peut être réalisée par l'emprunt de flexibilité à d'autres éléments, par exemple.
- La *cohérence* d'un système comprenant des incontrôlables doit être redéfinie en termes de contrôlabilité. Il sera cohérent s'il est contrôlable, dans le sens où les durées peuvent être adaptées pour obtenir une nouvelle solution, quelles que soient les durées prises à l'exécution par les intervalles incontrôlables.

8.2 A quoi servent les incontrôlables?

Si les incontrôlables posent problème, pourquoi les utiliser? Voici la réponse.

Les modèles temporels précédents ne sont pas réalistes dans le cadre de l'édition multimédia, car ils sont statiques. Il supposent que le système de présentation a le pouvoir de choisir une durée à l'intérieur de chaque intervalle de validité de chaque objet. Ce n'est pas le cas des intervalles incontrôlables. Or, les éléments incontrôlables sont indispensables dans notre contexte d'édition multimédia. Les incontrôlables proviennent de l'extérieur, de la nature entourant le système d'édition. Il en existe deux sortes.

La première concerne l'environnement informatique. Certains éléments sont continus, comme les vidéos. Lors de l'*accès via réseau* à une vidéo, les temps de chargement ne sont jamais garantis. Si le réseau est encombré ou que le CPU est surchargé, le lecteur d'un document veut le visualiser quand même. Pour cela, il faut pouvoir adapter dynamiquement certains paramètres, comme la diminution de la résolution ou du nombre d'images par seconde. D'autres éléments sont de caractère discret, comme les réponses à des requêtes adressées à des bases de données.

La seconde sorte, beaucoup plus importante, comprend les *interactions des utilisateurs*. Si un bouton est activable pendant 20 à 30 secondes, rien ne prédit à quel moment l'utilisateur l'activera, ni même s'il l'activera. Prenons l'exemple où un bonhomme doit traverser l'écran après que l'utilisateur ait enfoncé un bouton, et que l'ensemble doive durer 30 secondes. Les deux objets sont incontrôlables. Selon le moment où l'utilisateur enfonce le bouton, la vitesse du bonhomme sera différente.

8.3 Une première solution

Les incontrôlables sont indispensables et complexes. Il faut pouvoir résoudre les problèmes qu'ils soulèvent. Voici une première solution. C'est une approche partielle, pragmatique, qui résout seulement un nombre limité de cas bien précis. Considérons d'abord l'exemple simpliste de la figure 8.1. L'objet A est un intervalle contrôlable (un texte), tandis que l'objet B est incontrôlable (bouton à enfoncer par l'utilisateur). Quel que soit le temps t où l'utilisteur enfonce le bouton, il sera possible au système d'édition de fixer à ce moment la durée du texte A à t secondes. Dans tous les cas de figure, le système est adaptable: il est dit cohérent.

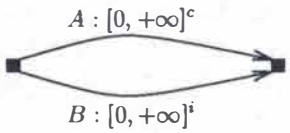


FIG. 8.1 – *Incontrôlabilité : système cohérent (cas simpliste)*

Prenons l'exemple un rien plus complexe de la figure 8.2. Un intervalle contrôlable B de durée $[b_1, b_2]$ dispose d'une certaine flexibilité $b_2 - b_1$ et est précédé d'un objet D dont la durée a été fixée. Un intervalle incontrôlable A de durée $[a_1, a_2]$ est suivi d'un objet C dont la durée est fixée aussi. Le système est formé de deux séquences mises en parallèles, appelées chaînes. Pour résoudre ce cas, l'idée est de compenser l'indéterminisme $a_2 - a_1$ de A par la flexibilité $b_2 - b_1$ de B. Le système sera cohérent sous deux conditions. La première, c'est que, $a_2 - a_1 < b_2 - b_1$. Autrement dit, il faut disposer de suffisamment de flexibilité sur B pour combler l'indéterminisme de A. La seconde condition est que la fin effective de A précède le

début de B : la durée à compenser ne sera connue qu'à la fin de A, et il faudra avoir fixé la durée compensée de B avant qu'il ne soit commencé. En effet, si B représente un bonhomme traversant l'écran, il faut connaître dès le début de sa présentation quelle sera sa vitesse.

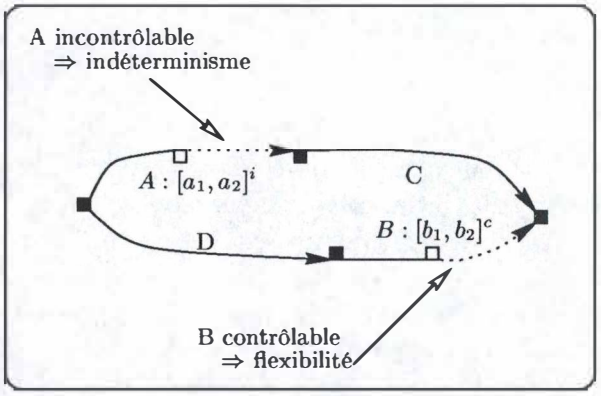


FIG. 8.2 – *Incontrôlabilité : système cohérent ? (Cas un peu moins simple)*

Appelons *chaîne* un chemin chaque sommet (sauf les deux extrémités) n'ont qu'un arc entrant et un arc sortant. La première idée de solution, reprise de [Layaïda96b], est la suivante. Nous considérons tous les couples de chaînes ayant même origine et même extrémité. Sur chaque boucle ainsi définie, nous étudions les chaînes C_1 et C_2 qui la forment. La cohérence est assurée s'il est possible de compenser les indéterminismes de l'une par les flexibilités de l'autre, et réciproquement. En réalité, il faut de plus assurer une réservation des portions de flexibilités accordées à chaque intervalle indéterministe. En outre, il faut que les instants de fin des intervalles incontrôlables compensés se situent avant le début de l'intervalle contrôlable auquel la flexibilité est empruntée, sinon la compensation ne pourra avoir lieu à temps lors de l'exécution.

8.4 Nouvelle cohérence : types de contrôlabilité

Une solution partielle n'est pas toujours satisfaisante : il faut des théories adaptées, de nouveaux concepts. En voici. Des recherches actives sont en cours pour régler le problème difficile des incontrôlables, comme [Fargier98] et [Fargier97]. La notion de cohérence est redéfinie en terme de contrôlabilité pour s'adapter aux intervalles incontrôlables. Lors de la vérification de cohérence, il restera des intervalles non réduits à une durée. Une fois les incontrôlables déterminés, le système est équivalent à un STP classique. Ceci mène aux trois nuances de cohérences suivantes :

1. *Contrôlabilité faible*, "$\forall situ, \exists solution$", la plus large des trois. Cette propriété est acquise lorsque, quelle que soit la valeur prise par les intervalles incontrôlables dans une situation donnée, le STP équivalent possède au moins une solution.
2. *Contrôlabilité forte*, "$\exists solution, \forall situation$", la plus précise des trois. Cette propriété est vérifiée lorsqu'une seule instanciation commune des durées est à la fois solution de tous les systèmes générables par les durées des incontrôlables, quelles que soit la valeur qui leur sera affectée.

3. *Contrôlabilité dynamique*, intermédiaire entre les deux précédentes, définie dans [Fargier97]. Elle différencie les instants "activés" (choisis par le système) des instants "reçus" (choisis par l'extérieur). Elle distingue les situations connues (durées fixées dans le passé) des situations encore inconnues (durées à fixer dans l'avenir). La contrôlabilité dynamique est vérifiée lorsque, pour chaque situation passée, il est possible de trouver une solution en choisissant des durées pour les instants activés à venir, quelle que soit la valeur des futurs instants reçus.

La cohérence forte est souvent trop forte. Elle exige qu'une seule solution satisfasse tous les STP générables par tout choix de durée des intervalles incontrôlables. S'il existe une seule instanciation des intervalles incontrôlables pour laquelle il n'existe pas de solution, le document sera rejeté globalement, même si la plupart de ses exécutions ne pose pas problème ! Elle n'est donc pas intéressante, même si son temps d'exécution polynômial l'est.

La cohérence faible est souvent trop faible. Elle accepte des documents qui ne devraient pas l'être. Des situations où une exigence est satisfaite mais pas une autre peuvent survenir, ce qui donne un document pour lequel la suite de la présentation ne sera plus solution du système. De plus, sa détection est NP-complète.

La cohérence dynamique est intermédiaire et nous intéresse au plus haut point. Elle assure que le système de présentation pourra toujours trouver une solution ultérieure à partir de toute situation présente. Malheureusement, sa détection n'est pas polynômiale : elle serait exponentielle au pire.

8.5 Nouvelles solutions

Nous avons présenté une façon partielle de régler le problème des incontrôlables, en tentant de compenser les incontrôlabilités par les flexibilités de chaînes parallèles. Une solution théorique plus complète devrait être trouvée via la notion de cohérence dynamique.

Les algorithmes de présentation utilisant la cohérence dynamique sont du type "jeu contre la nature". La nature, c'est ce qui choisit la durée des intervalles (incontrôlables) : ceux que le système n'a pas le droit de fixer lui-même. L'algorithme va devoir se battre contre la nature à travers le temps. A chaque étape, la nature peut fixer la durée d'intervalles incontrôlables. L'algorithme devra réagir de manière à ce que le système admette toujours une solution dans le futur. Il reste à inventer des algorithmes de nature réactive qui répondent à ces exigences.

Un des problèmes restants est l'incomplétude : certaines incohérences peuvent survenir dans certains cas, même si la cohérence dynamique est assurée, d'après [Fargier98, page 4]. De plus, le temps d'exécution est encore trop long. Sans doute faudra-t-il simplifier les données du problème jusqu'à obtention de systèmes réduits dont le temps d'exécution devient acceptable dans un processus d'édition. Les recherches sont en cours ...

Ce document touche à sa fin. Nous allons terminer par l'habituel chapitre consacré aux conclusions et perspectives à venir.

Chapitre 9

Pour aller plus loin: perspectives et conclusion

Nous voici arrivés en fin de ce mémoire. Nous allons présenter un résumé de ce qui a été fait tout au long de cet ouvrage. Nous continuerons en précisant les perspectives à venir qu'il serait intéressant d'aborder. Le présent mémoire est clôturé par quelques annexes, dont la dernière dresse un aperçu de ce que j'ai fait et appris au cours de mon stage et de la rédaction de ce mémoire.

9.1 Résumé

Il est temps de résumer ce que nous avons fait. Nous avons étudié l'édition de documents multimédia selon deux classes d'approches: les approches opérationnelles et les approches déclaratives (par contraintes). Ces dernières sont plus intéressantes du point de vue de l'auteur. Nous avons présenté les principes d'un éditeur fonctionnant selon cette approche (Madison) et le langage de description de documents utilisé (Madeus). Les contraintes utilisées dans les approches déclaratives fournissent la flexibilité des relations temporelles inter-objets. Les systèmes de contraintes utilisés sont limités dans un premier temps aux STP. Ils supposent deux phases supplémentaires au cours d'un processus d'édition : la vérification de cohérence et le formatage. Nous avons donc examiné la notion de cohérence et les principes de sa vérification. Ensuite, nous avons examiné un algorithme particulier s'acquitant de cette tâche: PC2, ainsi que les problèmes rencontrés lors de son implémentation. Enfin, nous avons examiné la notion d'intervalle incontrôlable. Il est indispensable de gérer ces intervalles dans l'édition temporelle multimédia, pour tenir compte notamment des interactions utilisateurs. Cela pose une série de problèmes que nous venons d'examiner au chapitre précédent.

9.2 Pour aller plus loin

Nous avons été confrontés à plusieurs limites.

La première, peu évoquée, concerne les relations causales. Ces dernières nécessitent de distinguer les instants prévus des instants effectifs, car un objet peut provoquer la terminaison d'un autre. Certains cas peuvent se ramener aux algorithmes étudiés. C'est le cas d'une relation "Par_min" entre deux objets contrôlables, qui est réductible à un objet composite les contenant. Mais il existe d'autres cas, et cette dimension reste majoritairement inexplorée,

alors que son utilité est évidente dans le cas d'interactions utilisateur ("Par_master" entre un bouton et le média que ce bouton arrête, par exemple).

La seconde concerne la gestion des intervalles incontrôlables. Il existe des solutions partielles temporaires, mais elles ne sont pas entièrement satisfaisantes. De nouveaux concepts sont apparus, mais la théorie est encore incomplète. Une nouvelle voie semble ouverte avec la notion de cohérence dynamique [Fargier97]. Certains proposent déjà des extensions à ce nouveau modèle théorique alors qu'il n'est même pas encore complet [Fargier98, page 4]. L'étude des algorithmes réactifs du style "combat contre la nature" pourront peut-être nous apporter des réponses. Peut-être la théorie des jeux ou les automates temporisés pourront-ils faire de même.

L'élément clé de notre environnement incrémental d'édition et de présentation de documents multimédia structurés réside principalement dans la performance des algorithmes de vérification de cohérence et de formatage. Le problème de vitesse d'exécution est l'un des principaux obstacles actuellement.

Pour conclure, de toute évidence, il reste beaucoup à faire pour améliorer la gestion des contraintes temporelles utilisées dans l'édition de documents multimédia structurés.

Annexe A

Approches d'édition temporelle

Voici un tableau récapitulatif comparant des éditeurs multimédia en utilisant les 15 critères définis dans la section 2.3.1 page 19. La légende se situe juste après :

num.	critère / outils édition	CMIFed	Isis	Firefly	Madeus	Mheg	Smil
1	obj : variété	0	0	0	0	0	0
2	obj : contôle (continu)	-	-	-	0	-	-
3	obj : interaction	0	?	?	+	+	+
4	obj : styles	-	-	-	+	-	-
5	obj : non prédictible	+	-	0	0	+	0
6	composition temp.	0	0	0	0	0	-
7	interactivité	+	0	0	0	+	0
8	facilité	0	+	+	+	?	?
9	création directe	-	+	+	+	?	?
10	indéterminisme	-	-	-	0	?	?
11	modif. locale	-	+	+	+	?	?
12	switch rapide	+	+	-	+	?	?
13	abstraction	+	+	-	+	?	?
14	modèles	-	-	-	-	?	?
15	multi lecture	0	-	-	0	?	?

Le tableau peut être lu de la manière suivante :

1. "-" : support peu ou mal visé
2. "+" : bon support fourni
3. "0" : support partiel
4. "?" : information non disponible

Les éditeurs comparés sont les suivants :

1. CMIFed. Il suit une approche opérationnelle par arbres hiérarchiques. Ses opérateurs temporels sont la séquence et le parallélisme.
2. Isis. Il suit une approche par contraintes. Ses pérateurs temporels sont les 4 relations de base d'Allen (meets, equals, starts, finishes) et les autres, obtenues par ajout d'un délai. Les relations de causalité (interruption) et les intervalles incontrôlables ne sont pas pris en compte.

3. Firefly. C'est une approche par contrainte. Ses opérateurs temporels sont les relations d'instants ("<", "=" et ">"). C'est le premier logiciel ayant tenté de prendre en compte les intervalles incontrôlables.
4. Madeus. Il suit un approche par contrainte. Ses opérateurs temporels sont une extension avec délais numériques des 13 relations d'Allen, auxquelles s'ajoutent des relations de causalité ("Par_min", "Par_max", "Par_master"). Les intervalles incontrôlables et les relations de causalité devraient être prises en compte prochainement.
5. MHEG et SMIL. Ce sont deux standards. La fin du tableau est vide, car aucun éditeur associé à ces langages n'a été étudié. Il en existe d'ailleurs très peu.

Annexe B

Définition du langage Madeus (DTD)

Le langage Madeus est défini dans le formalisme de XML 1.0. Il suffit de connaître XML pour être capable de définir un document Madeus. Notons que cette syntaxe ne donne évidemment pas la sémantique. Il faut savoir également que cette version n'est pas récente : elle date de novembre 1998. Voici donc la DTD de Madeus, dans les quelques pages qui suivent :

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

<!ELEMENT segment EMPTY>

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


<!--============================================================================-->
<!--============================================== Generique entities -->
<!--============================================================================-->

<!ENTITY % link_properties "HRef CDATA #IMPLIED">

<!ENTITY % position_properties "Left CDATA #IMPLIED
                               Top  CDATA #IMPLIED">

<!ENTITY % position_functions "(segment)">
<!ENTITY % color_functions "(segment)">
<!ENTITY % font_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
                             FileName CDATA #IMPLIED
                             BaseName CDATA #IMPLIED
                             Value CDATA #IMPLIED
                             BackGround CDATA #IMPLIED
                             Duration CDATA #IMPLIED
                             %link_properties;">

<!ENTITY % all_properties "%global_properties; %position_properties; %font_properties;">
<!ENTITY % objects "(Text | Video | Movie | Audio | Picture | Plugin | Unknown
            | Fictitious | External | Applet)">


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


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

<!ELEMENT Madeus (%objects; | Composite)>

<!ATTLIST Madeus
        Name CDATA #REQUIRED
        Version CDATA #REQUIRED>

<!--=========================================================================-->
<!--============================================================ Text object -->
<!--=========================================================================-->

<!ENTITY % text_properties "%position_properties;
                           %size_properties;
                           %color_properties;
                           %font_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;>

<!--=========================================================================-->
<!--============================================================ Video object -->
<!--=========================================================================-->

<!ENTITY % video_properties "%position_properties;
                             %size_properties;">

<!ENTITY % video_functions "%position_functions;">

<!ELEMENT Video (video_intervals?, relations?)>

<!ATTLIST Video %global_properties;
                %video_properties;>

<!ELEMENT video_intervals (video_interval+)>

<!ELEMENT video_interval (%video_functions;)*>

<!ATTLIST video_interval
      %global_properties;
          %video_properties;>

<!--=========================================================================-->
<!--============================================================ Audio object -->
<!--=========================================================================-->
```

```
<!ENTITY % audio_properties "%position_properties;
                             %size_properties;">

<!ENTITY % audio_functions "%position_functions;">

<!ELEMENT Audio (audio_intervals?, relations?)>

<!ATTLIST Audio %global_properties;
                %audio_properties;>

<!ELEMENT audio_intervals (audio_interval+)>

<!ELEMENT audio_interval (%audio_functions;)*>

<!ATTLIST audio_interval
        %global_properties;
          %audio_properties;>

<!--=========================================================================-->
<!--============================================================ Plugin object -->
<!--=========================================================================-->

<!ENTITY % plugin_properties "%position_properties;
                              %size_properties;">

<!ENTITY % plugin_functions "%position_functions;">

<!ELEMENT Plugin (plugin_intervals?, relations?)>

<!ATTLIST Plugin %global_properties;
                 %plugin_properties;>

<!ELEMENT plugin_intervals (plugin_interval+)>

<!ELEMENT plugin_interval (%plugin_functions;)*>

<!ATTLIST plugin_interval
        %global_properties;
          %plugin_properties;>

<!--=========================================================================-->
<!--============================================================ Applet object -->
<!--=========================================================================-->

<!ENTITY % applet_properties "%position_properties;
                              %size_properties;">

<!ENTITY % applet_functions "%position_functions;">

<!ELEMENT Applet (Param*)>

<!ATTLIST Applet %global_properties;
                 %applet_properties;>
```

```
<!ELEMENT Param EMPTY>
<!ATTLIST Param
        Name  CDATA #REQUIRED
        Value CDATA #REQUIRED>

<!--=========================================================================-->
<!--=================================================== Unknown object -->
<!--=========================================================================-->

<!ENTITY % unknown_properties "%position_properties;
                               %size_properties;">

<!ENTITY % unknown_functions "%position_functions;">

<!ELEMENT Unknown (unknown_intervals?, relations?)>

<!ATTLIST Unknown %global_properties;
                  %unknown_properties;>

<!ELEMENT unknown_intervals (unknown_interval+)>

<!ELEMENT unknown_interval (%unknown_functions;)*>

<!ATTLIST unknown_interval
      %global_properties;
          %unknown_properties;>

<!--=========================================================================-->
<!--=================================================== Fictitious object -->
<!--=========================================================================-->

<!ENTITY % fictitious_properties "%position_properties;
                                  %size_properties;">

<!ENTITY % fictitious_functions "%position_functions;">

<!ELEMENT Fictitious (fictitious_intervals?, relations?)>

<!ATTLIST Fictitious %global_properties;
                  %fictitious_properties;>

<!ELEMENT fictitious_intervals (fictitious_interval+)>

<!ELEMENT fictitious_interval (%fictitious_functions;)*>

<!ATTLIST fictitious_interval
        %global_properties;
          %fictitious_properties;>

<!--=========================================================================-->
<!--=================================================== Composite object -->
<!--=========================================================================-->
```

```
<!ELEMENT Composite (Default? , (%objects; | Composite)+, Relations?)>

<!ATTLIST Composite
          %global_properties;
          %position_properties;>

<!ELEMENT Default EMPTY>

<!ATTLIST Default
         %all_properties;>


<!--=========================================================================-->
<!--============================================================ 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 C

Expérience acquise

Cette annexe présente un rapide bilan des expériences acquises lors de mon stage et de la rédaction de ce mémoire.

Au cours du stage pratique à l'INRIA de Montbonnot-St-Martin, j'ai réalisé un travail en deux phases principales. La première a été une prise de contact avec l'environnement de travail. L'apprentissage du langage Java et de la philosophie orientée objet qui s'y rapporte a été relativement long, mais très instructif. Ensuite, il a fallu s'habituer à l'environnement logiciel : Linux, Emacs, cvs, Madison et Madeus. La seconde phase était de l'implémentation. Je me suis concentré sur la structure de graphe temporel et surtout sur l'algorithme PC2 (vérification de cohérence et formatage statique des documents Madeus hiérarchiques). Au cours du stage et de la rédaction du mémoire, j'ai beaucoup appris.

D'abord, je me suis familiarisé avec nombre d'outils et de méthodes en informatique. Je pense par exemple au travail de groupe sur les mêmes fichiers, grandement facilité par le gestionnaire de versions CVS. Mais ces outils ne remplaceront jamais une bonne communication entre les membres d'une équipe, comme celle que j'ai constatée sur place. Ils ne remplaceront jamais non plus une bonne démarche d'analyse. La structure des classes Java utilisée pour gérer un projet aussi complexe que Madison illustre une bonne mise en pratique des concepts liés aux méthodes et à la philosophie orientées objet.

Ensuite, j'ai évidemment découvert le domaine de l'édition multimédia. Je ne pense pas qu'il soit utile de rappeler cet aspect, car il a été assez longuement développé au cours de cet ouvrage ;-)

Enfin, j'ai acquis une expérience gratifiante lors de la rédaction du mémoire, notamment avec l'utilisation des logiciels et langages suivants : XFig (figures parsemant le texte), convert (transformation des .gif en .eps), vim (bibliographie en coloration syntaxique), bash et Perl (scripts corrigeant les index mal générés par LaTeX sous les options french et babel), ispell (correcteur lexical), WindowMaker (gestionnaire de fenêtres simple et très efficace), xdvi (visualisation des fichiers .dvi), dvips (transformation des fichiers .dvi en fichers postscript), LaTeX 2_ε (formatage impeccable) et LyX (interface graphique au-dessus de LaTeX 2_ε). Il n'est pas très sain de prôner une approche sans la mettre en pratique. J'ai défendu une approche par contraintes dans le domaine de l'édition de documents structurés. Or, LaTeX est un formateur de texte utilisant ces principes. En effet, il utilise la notion de boîtes rectangulaires à taille variable pour le formatage. La flexibilité nécessaire lui est fournie par la notion de "glue", qui permet notamment d'ajuster les espaces inter-mots en vue d'une justification gauche et droite du texte par rapport aux marges.

Pour terminer, j'ajoute que j'ai appris la patience et la persévérence requise par un travail de rédaction toujours assez long et difficile.

Bibliographie

[Acm93] Van Rossum G., Jansen J., Mullender K. and Bulterman D, *CMIFed : a presentation environment for portable hypermedia documents*, proceedings of the ACM multimedia conference, California (USA), 1993.

[Allen91] Allen J.F., *Maintaining knowledge about temporal intervals*, Communications of the ACM, vol 26, num 11, pp. 832–843, November 1983.

[Bes98] *Spécification hiérarchique de scénario temporel à base de contraintes*, F. BES, DEA d'informatique, université Joseph Fourier, juin 1998.

[Carcone97] L.Carcone, M.Jourdan, C. Roisin, *Présentation de documents multimédia basée sur les contraintes*, Workshop on Electronic Page Models, LAMPE, 22–23 septembre 1997.

[Def99] *Définition du terme multimedia*, site Webopedia, 1999, `http://webopedia.internet.com/TERM/m/multimedia.html`

[Faq99] *Multimedia Authoring Systems FAQ*, Version 2.22, 7 mars 1999, `http://www.tiac.net/users/jasiglar/mmasfaq.html`

[Fargier97] Thierry VIDAL, Hélène FARGIER, *Contingent durations in temporal CSPs: from consistency to controllabilities*, 4th Int. workshop on temporal representation and reasoning (TIME97), Daytona Beach (Florida), USA, May 10–11, 1997 — proceedings published by the IEEE.

[Fargier98] H. Fargier, M.Jourdan, N. Layaïda, T. Vidal, *Using Temporal Constraints Networks to manage temporal scenario of multimedia documents*, ECAI 98 Workshop on Spatial and Temporal Reasoning, Brighton (UK), August 1998.

[HyTime-1] *Information processing – Hypermedia/Time-based Structuring Language (HyTime)*, approved text, WG8, 2d edition, may 1997, `http://www.ornl.gov/sgml/wg8/docs/n1920/`

[HyTime-2] *HyTime et SGML*, `http://www.oasis-open.org/cover/hytime.html`

[HyTime-3] *HyTime user's group home page* `http://www.hytime.org/`

[HyTime-4] *HyTime tools list*, `http://www.hytime.org/tools/index.html`

[Jourdan97a] M. Jourdan, N. Layaïda, L. Sabry-Ismail, *Time Representation and Management in MADEUS: an authoring environment for multimedia documents*, ultimedia Computing and Networking 1997, M. Freeman, P. Jardetzki, H.M. Vin, SPIE 3020, San-Jose, February 1997.

[Jourdan97b] M. Jourdan, N. Layaïda, L. Sabry-Ismail, *MADEUS: an authoring environment for multimedia documents*, IEEE International Conference on Multimedia Computing and Systems, pp. 644–645, Ottawa (Canada), June 1997.

[Jourdan97c] M. Jourdan, C. Roisin, L. Tardif, *Visualization of constrainted-based Temporal Scenarios in a Multimedia Authoring tool*, num. 3284, INRIA, October 1997.

[Jourdan98b] M. jourdan, N. Layaïda, C. Roisin, *A survey on authoring techniques for temporal scenarios of multimedia documents*, vol. to be published in Handbook of multimedia, CRC Press, April 1998.

[Jourdan98c] M. jourdan, C. Roisin, L. Tardif, *Constraints Techniques for Authoring Multimedia Documents*, ECAI 98 Workshop on Constraints for artistic applications, Brighton (UK), August 1998.

[Jourdan98d] M. jourdan, N. Layaïda, C. Roisin, L. Sabry-Ismail, L. Tardif, *Madeus, an Authoring Environment for Interactive Multimedia Documents*, ACM Multimedia'98, September 1998.

[Jourdan98e] M. Jourdan, C. Roisin, L. Tardif, *Multiviews Interfaces for Multimedia Authoring Environments*, Proceedings of the 54th Conference on Multimedia Modeling, Lausanne, November 1998.

[Lamport94] Leslie Lamport, LaTeX, a document preparation system. User's guide and reference manual, Addison-Wesley, 2nd ed., New York, 1994.

[Lang99] *Authoring Languages Search Results*, `http://www.mcli.dist.maricopa.edu/authoring0/lang.html`

[Layaïda96a] N. Layaïda, *Issues in Temporal Representations of Multimedia Documents*, Workshop on Real Time Multimedia and the Web 96, Philipp Hoschka editor, World Wide Web Consortium, INRIA Sophia-Antipolis, October 1996.

[Layaïda96b] N. Layaïda, L. Sabry-Ismail, *Maintaining temporal consistency of multimedia documents using constraint networks*, multimedia computing and networking 1996, M. Freeman, P. Jardetzky, H. M. Vin editors,, Proc. SPIE 2667,pp. 124–135, San-José, USA, February 1996

[Layaïda97] Nabil Layaïda, *Madeus : Système d'édition et de présentation de documents multimédia structurés*, PhD Thesis from the University Joseph Fourier, Grenoble, France, June 1997.

[Meurisse99] M. Meurisse, *Visualisation des scénarios temporels dans Madeus*, mémoire de maîtrise, institut d'informatique, FUNDP Namur, septembre 1999.

[Mheg-1] *The MHEG Centre*, `http://www.mhegcentre.com/`

[Smil-1] *Synchronized multimedia : SMIL*, SMIL 1.0 Specification : W3C Recommendation (jun 1998), Working draft on SMIL successor : "SMIL Boston" (aug 1999), `http://www.w3.org/AudioVideo/`

[Smil-2] *Didacticiel SMIL écrit en SMIL*, `http://www.empirenet.com/~joseram/synchronization/synchronization.html`

[Smil-3] *SOJA : Smil Output in Java Applets*, may 1999, `http://www.helio.org/products/smil/`

[Smil-4] *Logiciel "SMIL Composer" de Sausage Software*, `http://sausage.com/supertoolz/toolz/stsmil.html`

[Standard-1] *Distributed Multimedia Survey: Standards*, `http://cuiwww.unige.ch/OSG/info/MultimediaInfo/mmsurvey/standards.html`

[Standard-2] Service Open Information Interchange de la commission Européenne, *What is OII*, juillet 1999, `http://www.echo.lu/oii/en/oii-info.html`

[Standard-3] *Multimedia/Hypermedia Interchange Standards*, section of the OII Standards and Specifications List, juillet 1999, `http://www2.echo.lu/oii/en/moving.html`

[Standard-4] *World Wide Web Consortium*, LE site de promotion de standards sur Internet, `http://www.w3.org/`

[Standard-5] *Standards liés à SGML (XML, SMIL, CSS, etc)*, site très informatif, incontournable en la matière, `http://www.oasis-open.org/cover/`

[Tardif97] L. Tardif, *Visualisation du scénario temporel d'un document multimédia*, Rapport de DEA, juin 1997.

[Tools-1] *Liste impressionnante d'outils d'édition de documents multimédia*, `http://www.davecentral.com/multiedit.html`

[Tools-2] *Macromedia Director : le site web*, `http://www.macromedia.com/software/director/`

[Tools-3] *Macromedia, plugin Shockwave*, `http://www.macromedia.com/shockwave/productinfo/`

[Tools-4] *Icon Author, the ultimate icon-based solution for multimedia and Internet-based training*, `http://www.asymetrix.com/products/iconauthor/`

[Tools-5] *Le site du logiciel ISIS*, `http://www.alphaworks.ibm.com/formula/isis`

Index