Gestion de données et de présentations multimédias par un SGBD à objets

Françoise Mocellin

Thèse

présentée par

Françoise Mocellin

pour obtenir le grade de Docteur

de l'Université Joseph Fourier - Grenoble 1

(arrêtés ministériels du 5 juillet 1984 et

du 30 Mars 1992)

Spécialité : **Informatique**

Gestion de données et de présentations multimédias par un SGBD à objets

---

Date de soutenance : 12 décembre 1997

Composition du jury :

Président :	Marie-France Bruandet
Rapporteurs :	Anne Doucet,
	Pierre Bazex
Examinateurs :	Andrzej Duda,
	Michel Adiba,
	Hervé Martin

Thèse préparée au sein du

Laboratoire Logiciels Systèmes Réseaux - IMAG

Remerciements

Je tiens à remercier :

Marie-France BRUANDET, *Professeur à l'Université Joseph Fourier et responsable du DEA Système d'Information qui me fait l'honneur de présider mon jury de thèse. Félicitations pour la réussite de ce DEA qui m'a permis de me spécialiser dans le domaine des systèmes d'information.*

Anne DOUCET, *Professeur à l'Université Pierre et Marie Curie de Paris pour son intérêt à mon travail, pour ses remarques et ses conseils qui m'ont permis d'améliorer mon manuscrit.*

Pierre BAZEX, *Professeur à l'Université Paul Sabatier de Toulouse, d'avoir bien voulu apporter son jugement sur ce travail.*

Andrzej DUDA, *Professeur à l'ENSIMAG, pour avoir gentiment accepté de participer à mon jury.*

Michel ADIBA, *Professeur à l'Université Joseph Fourier, pour m'avoir permis de continuer ses travaux dans ce domaine intéressant des bases de données multimédias, pour ses conseils tout au long de ces trois années.*

Hervé MARTIN, *Maître de Conférences à l'Université Pierre Mendès France, déjà quatre ans qu'il suit mon travail de près et avec intérêt. Merci pour sa confiance, ses critiques constructives et toutes ses idées qui apparaissent dans ma thèse. Merci pour tous ses conseils professionnels et personnels.*

Christine COLLET, *Maître de Conférences à l'Université Joseph Fourier, pour sa disponiblilité, son enthousiasme communicatif. Merci pour les nombreuses lectures de mon manuscrit accompagnées de remarques très intéressantes qui ont toujours contribuées à des améliorations notables.*

Merci à tous les membres de l'équipe STORM pour leur sympathie, leur aide dans le déroulement de mon doctorat, quatre années au cours desquelles j'aurai appris beaucoup et qui m'auront permises de m'épanouir dans mon travail et d'un point de vue personnel.

Un grand merci à Rafael pour son amitié, sa gentillesse et pour toute l'aide qu'il m'a apportée pour le développement de la version finale du prototype. Merci à José pour sa sympathie et pour notre agréable collaboration qui a donné lieu à la réalisation de l'atelier de constructions de présentations multimédias. Merci à Agnès pour m'avoir supporté tous les jours de la semaine durant ces trois années, j'aurai gagné une nouvelle amie.

Merci à tous mes amis qui, par leur présence, embéllissent ma vie.

Et un grand merci tout spécial à ma famille pour être toujours si présente près de moi dans les meilleurs comme dans les moins bons moments de ma vie.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte et motivation

Marier le son, les images photographiques et la vidéo, c'est le rêve des informaticiens et le but du multimédia. Mais le terme multimédia est souvent utilisé à tort et à travers. Nous pouvons tout de même donner une première définition d'ordre technique de ce concept. Les ordinateurs ont, au départ, été conçus afin de manipuler de l'information de type alphanumérique, texte et chiffres. Mais ces machines sont, maintenant, capables de manipuler des données de types différents, statiques et dynamiques. Ainsi, la définition la plus communément admise pour qualifier un système multimédia reste celle d'un système capable de manipuler au moins un des types d'information suivants : le son, l'image fixe de qualité photographique et l'image vidéo animée. Le multimédia fait, en général, référence aux données telles que la vidéo, l'image, le texte, l'audio. Ces nouveaux types de données diffèrent des données traditionnelles, l'information contenue est plus importante et plus riche. Il va falloir apprendre à l'interpréter et à la présenter. En effet, le multimédia trouve sa place dans de nombreuses applications (bureautique, médecine, éducation, banques, etc.).

De façon plus concrète, mais cependant réductrice, on pourrait définir un système multimédia complet comme étant la réunion en une seule machine d'un ordinateur, d'une télévision, d'un magnétoscope, d'une chaîne haute-fidélité, d'un répondeur téléphonique, d'un fax, et de tous les outils permettant de manipuler et de mixer images, son et données sous forme numérique. Le multimédia est en fait le moyen naturel d'étendre les fonctionnalités d'une machine en se fondant sur le fait que l'homme, avec ses cinq sens, est par nature lui-aussi multimédia.

1.1.1 Les nouvelles technologies

De nombreuses technologies sont apparues et ont permis l'essor du multimédia. Il s'agit bien sûr du développement des réseaux comme Internet ou ATM capables de véhiculer de l'information composée de données multimédias. Ils fournissent ainsi le support pour le développement des autoroutes de l'information et d'une nouvelle génération d'applications engendrée par le World-Wide Web (WWW).

Le World-Wide Web est un système d'information hypermédia sur le réseau internet. Il offre aux utilisateurs un outil efficace pour accéder à une grande variété de documents de façon très simple. Grâce à des interfaces clientes conviviales, le projet WWW a changé la façon de voir et de créer l'information des utilisateurs; il a créé le premier réseau hypermédia réparti. Un document hypertexte est un fichier de texte normal avec une différence importante, il contient dans son texte des "liens" vers d'autres parties du document lui-même, soit vers d'autres documents. Un document hypermédia est un hypertexte avec la différence que les liens peuvent référencer également des fichiers sons, images ou vidéo.

Cependant, le monde du Web ne permet un véritable échange d'informations structurées et facilement contrôlables que si ces informations sont structurées en base de données. En effet, les bases de données offrent un certain nombre d'avantages indéniables quant à la mise à jour des informations, quant à la structuration, quant aux outils permettant de gérer ces informations. On comprend bien l'intérêt d'une base de données quand on utilise des logiciels de réservation de billets de train ou lorsqu'on utilise le minitel pour rechercher un correspondant. Mais l'utilisation d'une base de données sur le Web ne se limite pas à la saisie de formulaires et à la restitution de données présentes dans la base, elle doit offrir des possibilités pour gérer les données multimédias et permettre la construction d'applications multimédias.

Les bases de données prennent ainsi leur essor dans le domaine du multimédia, elles permettent de manipuler les données multimédias. A l'heure actuelle, de nombreux SGBD multimédias relationnels comme *Sybase*, *UniSQL* ou orienté-objet comme O_2, *ObjectStore* supportent les BLOB (Binary Large Object) pour les données multimédias. Un BLOB est non typé, long et avec un champ de longueur variable utilisé pour stocker des données multimédias dans la base. Le SGBD stocke la donnée multimédia comme une donnée non typée et délivre simplement des blocs de données à l'application qui la demande.

La technologie orienté-objet est souvent utilisée pour le développement des SGBD multimédias. L'approche objet fournit un cadre de travail pour définir des types de données définis par l'utilisateur et la possibilité de supporter des relations complexes dans une base de données à objets [DDB91, CC96a]. Les techniques d'encapsulation, d'héritage et de classes imbriquées permettent un ensemble de standards pour des fonctionnalités à définir et à étendre pour les différentes instances spécifiques des données multimédias. Ces caractéristiques permettent aux applications d'introduire, inclure et gérer des données multimédias.

1.1.2 Les applications multimédias

Le multimédia a pris un essor important dans le monde informatique, car il trouve sa place dans de nombreuses applications. De nombreux secteurs utilisent le multimédia dans leurs applications. Il s'agit de la bureautique, médecine, géographie, éducation, formation continue, météo, banques, agences de voyages, publicité, courrier électronique, CAO, CFAO, ventes sur catalogues électroniques, vidéo domestique, immobilier, bibliothèques, droit, informations touristiques, information par les journeaux ou les revues, etc. Cette liste est longue et incomplète, nous constatons ainsi que le multimédia ne manque pas d'applications. Détaillons la spécificité de certaines applications :

- **Gestion de documents multimédias**

  Cette application est une des plus naturelles dans le domaine des bases de données multimédias. Elle est intéressante pour de nombreux domaines d'applications comme la documentation technique pour la maintenance de produits, l'éducation, les systèmes d'information géographique, le téléservice, etc. [KRRK93, CACS94]. Les travaux de [LSI96a, JLSI97] proposent un modèle de documents multimédias structurés MADEUS. Avec cet outil, ils travaillent sur la conception d'un environnement d'édition de documents multimédias avec pour objectifs : (1) la richesse d'expression pour le placement spatial et l'ordonnancement temporel; (2) l'interactivité et la convivialité de la phase d'édition; (3) la portabilité des présentations et l'interchangeabilité des documents.

- **Les systèmes de courrier électronique multimédia**

  Ils sont une forme avancée des systèmes de courrier électronique classique. Ils intègrent différentes applications comme l'édition de données multimédias et le "mail" vocal [TRR94]. Ce type d'environnements de communication peut être

bénéfique pour l'utilisation d'un système de stockage multimédia qui servirait à répertorier les messages multimédias du réseau.

- **Un magazine électronique multimédia**

  Un magazine électronique multimédia, appelé "MultiMedia forum" [Sa94a] a été développé au sein du GMD-IPSI. Il s'agit d'un environnement d'édition et de lecture où les clients peuvent accéder à l'information. Les fonctionnalités supportées par ce prototype peuvent être groupées en trois principales fonctions : (1) l'*importation de l'information* qui couvre la création et l'acquisition de l'information des auteurs ou éditeurs; (2) le *traitement de l'information* qui consiste à stocker, indexer, rechercher et manipuler les documents; (3) l'*exportation de l'information* exporte l'information vers d'autres environnements et la distribution de documents multimédias aux utilisateurs.

- **Un service de vidéo-sur-demande**

  Un service de vidéo-sur-demande offre la possibilité d'accéder à des données vidéo en réponse à des requêtes portant sur le contenu ou des descripteurs externes (par exemple, sélection sur les acteurs, le titre des films, ou la musique, etc.). Ce service est d'actualité dans le domaine du divertissement comme la télévision. Le propos du système VVB (Virtual Video Browser) [LAFG93] est justement d'offrir une application de vidéo-sur-demande exploitant une base de données vidéo.

- **Un SGBD multimédia pour une application de nouvelles-sur-demande**

  [OSEMV95] propose la conception d'un SGBD multimédia pour réaliser un système d'information multimédia pour des nouvelles-sur-demande distribuées. L'application de "nouvelles-sur-demande" utilise les services des réseaux pour délivrer des articles aux souscripteurs sous forme de documents multimédias. Différents journalistes insèrent des articles dans la base qui sont ensuite accessibles par les utilisateurs. Le SGBD multimédia propose une aide visuelle pour l'interrogation. Cette interface de visualisation de requêtes produit trois principales fonctionnalités pour les utilisateurs : *présentation*, *navigation* et *recherche de nouvelles*.

1.2 Problématique et objectifs de notre travail

Toutes ces nouvelles technologies et applications permettent aux utilisateurs d'accéder à ces nouvelles données multimédias qui rendent plus attractive l'utilisation de l'outil informatique. Il est nécessaire de leur offrir des systèmes conviviaux et simples d'utilisation permettant de mélanger les données multimédias aux données traditionnelles.

Le but de notre travail est de pouvoir intégrer le multimédia dans un Système de Gestion de Bases de Données pour qu'il puisse manipuler ces différents types de données et développer des applications pour tous types d'utilisateurs. Nous nous plaçons dans le cadre des bases de données à objets, puisque la technologie orienté-objet semble la plus adaptée aux données multimédias. Elle présente certains avantages par rapport au relationnel : richesse du modèle de données (e.g. héritage, objets complexes, méthodes), homogénéité avec les logiciels de construction d'interface qui utilisent une approche objet, extensibilité du langage de requêtes au travers de l'utilisation de méthodes et possibilité de définition de composants réutilisables.

L'inclusion du multimédia dans une base de données a un impact important dans sa conception, ses caractéristiques et ses fonctions. Si la base de données permet seulement le stockage des données multimédias pour leur délivrance, alors un serveur de fichiers couplé à des capacités de stocker des pointeurs, des noms de fichiers ou des identificateurs d'objets dans la base est suffisant. L'utilisation de la connaissance de la sémantique des différents médias comme la possibilité d'indexation, de recherche et d'information relative est vraiment une fonction ajoutée à la valeur d'un système de bases de données.

Tous ces nouveaux besoins dans le développement de bases de données multimédias ont révélé un certain nombre de problèmes dans la mise en œuvre des SGBD. Au niveau de l'interface du SGBD, il est difficile de dessiner la frontière entre les services généraux du SGBD et les services spécifiques aux applications. Par exemple, il est difficile de savoir si le SGBD doit permettre de stocker et de jouer des présentations multimédias, ces concepts étant très proches de l'interface qui se charge de tout ce qui sera perçu par l'utilisateur. Nous pouvons nous demander si le SGBD doit produire ses propres éditeurs pour les données multimédias ou si il doit juste produire les données dans une forme adaptée aux composants externes. Ces outils peuvent très bien se situer au niveau interface, mais le SGBD ne serait alors qu'un système de stockage de données multimédias. Nous allons maintenant définir clairement ce que sont pour nous les nouveaux besoins des systèmes de bases de données multimédias, ainsi que

leurs fonctionnalités.

1.2.1 Besoins des Bases de Données multimédias

Les besoins des bases de données multimédias sont répartis en cinq grands thèmes : (1) le stockage des données multimédias; (2) la modélisation de la structure des données multimédias qui nécessite des mécanismes d'indexation et une modélisation sémantique et cohérente des abstractions; (3) la modélisation de présentations multimédias qui composent des objets multimédias en spécifiant la synchronisation entre eux, parallèle ou séquentielle; (4) l'interrogation des objets multimédias sur des aspects descriptifs, temporels ou de synchronisation; et enfin (5) l'interaction avec l'utilisateur.

- *Stockage des données multimédias*

  Le stockage des données multimédias requiert une grande quantité de ressources en terme de capacité de stockage. Par exemple, une vidéo durant une heure et demi nécessite un giga byte pour être stockée, même si elle est fortement compressée [LG91a, KL96]. Et si la vidéo n'est pas compressée le même objet requiert cent giga bytes.

  Les gestionnaires de stockage traditionnels ne sont pas adaptés aux besoins de stockage de telles données multimédias [Chu96]. Ainsi il est nécessaire de définir de nouveaux systèmes de stockage comme un gestionnaire de stockage hiérarchique ou des serveurs de médias continus [AH91, Ran93, ORA96].

- *Modélisation de la structure des données multimédias*

  Un modèle de données doit permettre de manipuler la structure des données en accord avec son type de média. Par exemple, la structure hiérarchique d'une vidéo est formée de séquences, scènes et plans. Une séquence est un ensemble de scènes et chaque scène est elle-même composée d'une liste de plans.

  La sémantique des données doit être modélisée pour permettre des recherches sur le contenu, ainsi que sur l'information textuelle associés aux données multimédias. Il est nécessaire de fournir des mécanismes d'indexation pour ces données, il faut pouvoir leur associer une information pertinente dans le but de les interroger. Le processus d'indexation peut être automatique ou manuel.

- *Modélisation des présentations multimédias*

  Un modèle de données permet la modélisation de la composition des objets et la synchronisation qui exprime des relations temporelles et spatiales entre les

objets. Les modèles temporels basés sur des opérateurs d'intervalles [All83] sont souvent utilisés pour modéliser les relations séquentielles et parallèles entre les objets. Ces notions sont étrangères aux modèles de données conventionnels, c'est pourquoi certaines extensions leur sont nécessaires pour concevoir de nouveaux modèles de données répondant aux besoins des applications.

Une présentation multimédia compose différents médias suivant une certaine synchronisation (parallèle ou séquentielle) dans le but de les présenter à un utilisateur. Chaque média possède ces propres aspects temporels et des caractéristiques spatiales qu'il faut prendre en compte pour le présenter.

- *Interrogation des objets multimédias*

  Un SGBD multimédia doit proposer des outils pour des recherches sur les aspects structurels, temporels ou sur le contenu des données multimédias. Les applications multimédias demandent une grande quantité de données, aussi il est nécessaire de développer de nouvelles techniques pour l'extraction des données.

  La navigation classique n'est pas suffisante pour accéder aux données, il faut définir des langages de requêtes appropriés. La recherche de certaines données multimédias requiert l'utilisation de techniques d'indexation évoluées.

- *Interaction avec l'utilisateur*

  Les présentations multimédias sont liées à la notion d'opérateur humain. L'utilisateur doit pouvoir interargir au cours du déroulement de la présentation. Ainsi, on obtient différents scénarios possibles pour l'exécution d'une présentation. Une interaction avec l'utilisateur est en fait composée d'un événement qui déclenche une action, celle-ci décrivant le comportement lié à cette interaction.

Notre approche apporte plutôt des solutions au trois derniers niveaux de cette classification. Au cours de cette thèse, nous ne nous intéresserons pas à apporter une solution au stockage des données multimédias, ce problème se situant plus au niveau système. Notre étude se situe plus dans le cadre de la modélisation des présentations, nous cherchons à étendre les modèles de données traditionnels pour prendre en compte les données multimédias et pouvoir ainsi construire, manipuler, interroger et jouer des présentations multimédias. Enfin, cette thèse apporte une solution à la modélisation et à l'exécution des différents comportements que l'on peut associer à une présentation permettant d'intégrer l'interaction avec l'utilisateur.

1.2.2 Fonctionnalités d'un SGBD multimédia

Nous pensons que le SGBD doit permettre de stocker, manipuler, interroger et jouer des présentations. Il n'est pas un simple serveur d'objets multimédias. Il doit prendre en compte les différents besoins que nous venons de décrire.

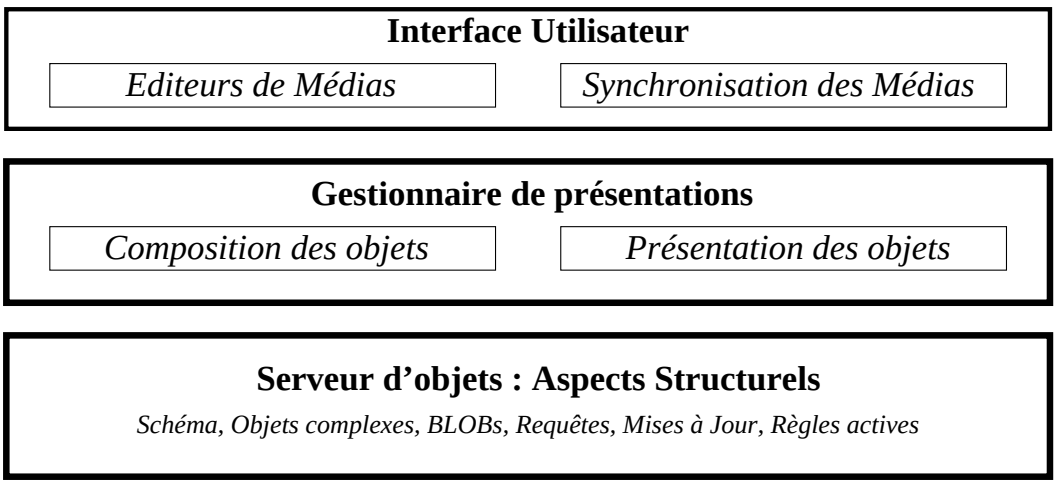

FIG. 1.1 – *L'architecture fonctionnelle d'un SGBD multimédia*

Au lieu de redéfinir un nouveau SGBD, nous proposons d'étendre les SGBD à objets existants et d'utiliser, si possible, leurs outils pour gérer les objets mutlimédias. Nous utilisons un modèle de données objet pour modéliser la structure spécifique de chaque média et modéliser les présentations multimédias. L'architecture fonctionnelle du SGBD multimédia, présentée sur la figure 1.1, se compose de quatre niveaux :

- L'*interface utilisateur* doit être composée d'outils pour l'édition des données multimédias et pour créer et jouer les présentations multimédias en accord avec leurs contraintes de synchronisation. Elle doit enfin fournir des services pour l'interrogation des données multimédias et des présentations multimédias.

- Le *gestionnaire de présentations multimédias* modélise les relations entre les objets pour pouvoir créer, rechercher et exécuter des présentations multimédias complexes. Elles sont composées de différents objets avec des contraintes spatiales et temporelles.

- Le *serveur d'objets* prend en charge les aspects classiques d'un SGBD (gestion d'un schéma, stockage d'objets, requêtes, transactions, concurrence, reprise).

Mais il doit également offrir des possibilités pour gérer chaque type de média avec des techniques de stockage adaptées à chacun.

En résumé, les principales fonctionnalités que doit posséder un SGBD multimédia sont :

1. être un support pour le stockage des données multimédias (capacités de stocker un très grand nombre d'objets multimédias, de gérer les différents formats des données, etc.),

2. intégrer au modèle de données les aspects temporels et spatiaux liés aux données multimédias : la composition temporelle est le processus qui assure que chaque élément apparaît à un temps voulu et joue pendant une certaine période, alors que la composition spatiale décrit le processus de rassemblement d'objets multimédias sur un écran à un certain moment du temps,

3. modéliser à la fois la composition des objets, la synchronisation entre les objets et l'interaction avec l'utilisateur : modélisation de présentations,

4. modéliser la sémantique des données multimédias pour permettre des recherches sur le contenu (le modèle de données doit permettre de manipuler la structure des données en accord avec son type de média), et fournir des techniques d'indexation évoluées pour interroger les données multimédias (un index est construit pour pouvoir associer une certaine information pertinente avec des morceaux de données),

1.2.3 Les applications

L'utilisation d'un SGBD multimédia trouve sa place dans de nombreuses applications. Par exemple, dans le domaine de la médecine, il peut être utilisé pour le stockage d'informations concernant des interventions chirurgicales. Il s'agit de vidéos filmées au cours de ces interventions, mais aussi d'annotations vocales et textuelles sur le déroulement de l'opération. Une fois ces informations stockées dans la base, il faut bien sûr offrir à l'utilisateur des facilités pour les rechercher et les présenter. Le SGBD doit donc offrir la possibilité de composer toutes ces informations entre elles pour obtenir une présentation pour l'utilisateur. On obtiendra, par exemple, une présentation parallèle d'une vidéo de l'intervention, d'un commentaire sonore du chirurgien et d'une séquence de textes décrivant les étapes de l'intervention (sous forme d'un diaporama).

De plus, il doit proposer des fonctionnalités d'interrogation permettant de retrouver ces présentations selon différents critères et l'utilisateur pourra visualiser à nouveau la présentation de l'intervention. Les critères de recherche doivent être variés, ils peuvent se baser sur un mot-clé ou sur la présence d'un séquence vidéo, etc.

L'utilisation d'un SGBD multimédia parait aussi souhaitable dans le cas de services de vidéos-sur-demande ou de nouvelles-sur-demande. Si les vidéos et les nouvelles sont stockées dans une base de données, on peut offrir à l'utilisateur des possibilités d'interroger ces données selon différents critères. Une fois retrouvée, l'information doit aussi être présentée à l'utilisateur.

De nombreuses applications multimédias nécessitent d'utiliser un SGBD. Il apparaît clairement que le concept de présentation multimédia est important et utile pour présenter les données multimédias à l'utilisateur. De même, les possibilités d'interrogation offertes par un SGBD sont capitales dans une application qui a besoin de rechercher toutes sortes d'informations.

1.3 Démarche et contribution de notre travail

1.3.1 Manipulation de présentations multimédias dans un SGBD à objets

Nous nous intéressons plus particulièrement à la modélisation de présentations multimédias et nous nous basons sur le modèle STORM proposé par Michel Adiba [Adi95, Adi96, AM97]. Ce modèle étudie les problèmes suivants : (1) la modélisation et la gestion des données basées sur le temps et (2) les possibilités, au niveau des langages et des interfaces pour la construction, la recherche et la mise à jour de données multimédias (Texte, Image, Audio, Vidéo). Il apporte une solution originale à certains de ces problèmes en proposant une approche à objets pour les données multimédias. Il permet de décrire des présentations séquentielles ou parallèles, de les construire, les jouer et les interroger.

Une présentation inclut : (1) les objets à présenter, (2) les contraintes temporelles à leur appliquer (délai, durée constituant l'*Ombre Temporelle* de l'objet) et (3) les contraintes de synchronisation et de transition entre objets (l'un après l'autre, en parallèle, etc.). La prise en compte de la synchronisation, en considérant des intervalles temporels, a permis d'utiliser les relations d'Allen pour traiter dans une présentation

plusieurs objets. Cette approche consiste à considérer chaque présentation comme un objet sur lequel des opérations spécifiques peuvent être définies (interrogation, mise à jour, et exécution). Il est possible d'associer diverses présentations à un même objet : par exemple une image peut être présentée seule pendant deux minutes, ou bien associée (et synchronisée) avec un commentaire audio dans une autre présentation. La notion d'*Ombre Temporelle* est centrale dans cette approche car elle permet de caractériser plusieurs aspects temporels pour un objet quand celui-ci fait partie de différentes présentations.

Les différentes présentations ainsi construites sont stockées sous forme d'objets dans la base. Pour l'implantation de ce modèle, nous proposons une **bibliothèque de classes** prédéfinies et réutilisables, qui peut être utilisée dans différentes applications multimédias. Elle peut être étendue suivant les besoins de l'utilisateur. Les objets de ces classes correspondent à des présentations d'objets monomédias où à des présentations plus complexes composées d'images, de vidéos, de sons synchronisés entre eux. Il est alors facile de rechercher ces objets pour les présenter à nouveau, par exemple, rechercher toutes les présentations où est présentée une vidéo de montagne. Nous donnons la possibilité d'interroger les présentations multimédias sur leurs aspects structurels, temporels et de synchronisation.

1.3.2 Modélisation des comportements d'une présentation

Le modèle de présentations décrit précédemment est centré sur la notion d'*Ombre Temporelle* qui définit les valeurs temporelles associées à un objet dans le temps. En effet, le déroulement d'une présentation est prévisible dans le temps. La liberté d'exécution est insuffisante, il faut pouvoir interagir avec la présentation, pouvoir modifier son comportement par défaut.

Le but du **modèle de comportements d'une présentation** est de rajouter une nouvelle dimension aux objets STORM pour pouvoir leur associer un comportement particulier au cours de leur présentation. L'ensemble des comportements lié à la présentation constitue l'*Ombre Comportementale* des présentations. Le déroulement de la présentation devient imprévisible et dépend des événements qui se produisent et qui déclenchent une suite d'actions. Ces réactions à différents événements augmentent la dynamique de l'exécution de la présentation.

On appelle *comportement d'une présentation d'un objet*, l'ensemble des actions qu'il est susceptible d'entreprendre suite à des événements lors de sa présentation. Par

exemple, le déroulement de la présentation va être influencé par des événements de l'utilisateur ou par des événements provenant de la présentation d'autres objets. Ainsi on associe à un objet une *Ombre Temporelle* pour définir ses aspects temporels, et une *Ombre Comportementale* pour ses aspects comportementaux. Nous verrons que l'*Ombre Comportementale* joue deux rôles principaux : (1) changer le déroulement prédéfini de la présentation en un déroulement suivant différentes alternatives qui correspondent à différents comportements de la présentation et (2) segmenter la présentation en associant des actions à chaque segment. Notre modèle de comportements propose la modélisation de l'*Ombre Comportementale* qui est définie par un ensemble de comportements. Chaque comportement est modélisé par un type d'événement (déclencheur du comportement), un intervalle de validité et une liste d'actions à exécuter.

Nous avons choisi d'implanter les différents comportements sous forme d'*objets scénarios* dans un but d'homogénéité de notre approche. Un objet scénario associe un comportement (ou une réaction) particulier à un objet d'une présentation (une image, une vidéo, etc.). Cette approche objet des comportements permet la *réutilisabilité*, la *mise à jour facile* et aussi la *coopération entre auteurs*. Un certain nombre de comportements sont prédéfinis et proposés au constructeur de la présentation. Nous proposons à l'utilisateur une **bibliothèque de classes de comportements** prédéfinie et réutilisable. Cette modélisation nous permet aussi d'interroger les comportements associés à une présentation à l'aide d'un langage tel `OQL`.

Au niveau exécution, nous montrons comment nous pouvons utiliser la technologie des SGBD actif [Col96] dans le but d'exécuter nos présentations multimédias actives temporelles. Les règles actives (ou règles Evénement-Condition-Action) vont nous permettre d'exécuter les différents comportements liés aux objets de la présentation. Nous avons constaté que la sémantique des comportements est très proche de celle d'une règle active E-C-A (Evénement-Condition-Action). Ainsi, les différents *objets scénarios* sont traduits en règles actives pour pouvoir être joués au cours de la présentation. En effet, les règles actives permettent de réagir à un événement et d'exécuter différentes actions suivant une certaine condition.

1.3.3 Extensions d'un SGBD à objets

Le dernier point important abordé dans cette thèse est la réalisation des concepts développés dans nos modèles. Nous avons en effet développé un prototype au dessus du SGBD O_2 [BDK92, AC93]. Ce prototype réalise notre définition et conception d'un SGBD multimédia, il montre ainsi la faisabilité des concepts énoncés au cours de cette

thèse [MMA96, Moc96a, Moc96b, ALMM97].

L'avantage d'étendre un *SGBD à objets* existant est de pouvoir utiliser toutes ses fonctionnalités. Les extensions multimédias correspondent à des bibliothèques de classes prédéfinies permettant de construire des présentations. Nous avons aussi réalisé le **moteur d'exécution** pour jouer ces présentations. Pour l'interrogation, nous utilisons le langage `OQL` et nous offrons certaines fonctions supplémentaires pour permettre une recherche sur les aspects de synchronisation entre les objets d'une présentation.

Nos extensions multimédias seront ensuite implantées au-dessus d'un *SGBD actif* puisque nous voulons utiliser les règles actives pour exécuter nos comportements. Le SGBD O_2 utilisé peut devenir actif grâce au système NAOS qui permet la définition et l'exécution de règles actives pour les applications O_2 [CCS94, Col97]. NAOS a, en effet, été développé en considérant le SGBD O_2 comme noyau de gestion des données et d'exécution des applications. Il s'intègre dans l'architecture modulaire de ce système et constitue donc un nouveau composant utilisable pour le développement des applications, à côté des autres composants élémentaires que sont O_2C, l'interface C++ et O_2SQL.

1.4 Organisation du document

- Le chapitre 2 présente un état de l'art des différentes approches s'intéressant au développement des nouveaux systèmes multimédias. Nous étudions leur façon d'aborder et de concevoir les différentes fonctionnalités que doit posséder un système de gestion de bases de données multimédias, qu'il s'agisse des aspects temporels et spatiaux liés aux données multimédias, de la modélisation ou la recherche de l'information multimédia ou enfin de la modélisation de l'interactivité avec l'utilisateur.

- Le chapitre 3 présente le modèle de présentations multimédias STORM [Adi95, Adi96, AM97]. Ce modèle s'appuie sur une approche objet pour contruire des présentations séquentielles ou parallèles qui composent différents médias entre eux. Il propose d'associer des aspects temporels aux données multimédias pour les présenter et de stocker les présentations sous forme d'objets dans la base. Enfin, nous montrerons comment un utilisateur peut créer une présentation STORM.

- Le chapitre 4 décrit notre modèle de comportements d'une présentation multimédia, son but étant d'augmenter la dynamique dans son déroulement. Nous associons différents comportements à une présentation. Un comportement est constitué d'un

ensemble d'actions déclenchées suite à un événement lors de l'exécution de la présentation. Notre modèle propose la modélisation de l'*Ombre Comportementale* d'une présentation définissant l'ensemble des comportements de la présentation.

- Le chapitre 5 présente notre réalisation d'un SGBD multimédia à partir des concepts des deux modèles présentés aux chapitres 3 et 4. Nous décrivons l'architecture de notre prototype qui a été développé au dessus du SGBD O_2. Nous proposons des extensions multimédias à ce SGBD à objets sous forme de bibliothèques de classes dans le but de pouvoir construire, jouer et interroger des présentations multimédias. Nous présentons aussi notre bibliothèque de classes de comportements dont les intances permettent de créer différents comportements et comment ceux-ci seront exécutés à l'aide de règles actives proposées par la technologie des SGBD actifs.

- Le chapitre 6 conclut ce document en mettant l'accent sur les apports de notre travail et en donnant quelques perspectives pour des recherches futures comme la définition des aspects spatiaux, la construction de "présentations virtuelles" ou l'indexation des présentations, etc.

Remarque - Les chapitres 3 et 4 apportent plutôt des solutions pour la manipulation de présentations multimédias au **niveau modélisation**. Alors que le chapitre 5 se situe plus au **niveau réalisation**.

Chapitre 2

Etat de l'art

Dans l'introduction, nous avons énoncé les nouvelles technologies qui rentrent en compte dans le développement du multimédia. Ensuite, nous avons défini les différents besoins des bases de données pour intégrer et manipuler les données multimédias. En nous basant sur cette étude, ce chapitre est consacré à un état de l'art sur les différentes approches et les problèmes qui se posent pour le développement des *Systèmes de Gestion de Bases de Données Multimédias*. Il est le résultat d'une synthèse d'un grand nombre de travaux s'intéressant aux nouvelles données multimédias et leur intégration dans les systèmes pour former les nouveaux SGBD.

Certains outils multimédias comme les systèmes auteurs sont couramment utilisés pour développer des applications multimédias. Mais nous allons vers la définition de systèmes multimédias complexes. Ils prennent en compte les aspects temporels des données multimédias, les synchronisent entre elles en séquence ou en parallèle pour les présenter. Ils offrent une modélisation de ces données pour pouvoir les manipuler, les interroger, effectuer des recherches basées sur leur contenu, etc. Toutes ces caractéristiques doivent être intégrées aux SGBD multimédias. Ceux-ci deviennent aussi plus interactifs, dans le sens où ils doivent prendre en compte l'interaction avec l'utilisateur lors de la présentation des données.

L'étude de l'intégration des aspects interactifs dans les SGBD nous a conduit à nous intéresser à la nouvelle technologie des SGBD actifs. En effet, ces systèmes sont capables de réagir à des événements en prenant des décisions et en exécutant des actions [CHR96]. La plupart du temps, les SGBD actifs utilisent la notion de règle active de la forme *Evénement-Condition-Action*. La sémantique d'une règle E-C-A est proche de celle d'une interaction qui réagit à un événement utilisateur pour exécuter

certaines actions. Ainsi, nous avons étudié la technologie des SGBD actifs dans le but de l'intégrer à celle des SGBD multimédias pour augmenter la dynamique d'une présentation multimédia, comme par exemple, prendre en compte l'interaction avec l'utilisateur.

Ce chapitre est une présentation du domaine des SGBD multimédias. Son objectif est de définir le domaine, de donner les définitions et références nécessaires à la compréhension de la suite du document, et donc, en définitive, d'esquisser le cadre de notre étude.

La section 2.1 présente les différentes types de données multimédias. La section 2.2 présente les caractéristiques et les fonctionnalités des systèmes auteurs qui permettent de développer des applications multimédias. La section 2.3 propose, à partir de différentes approches, les différentes fonctionnalités que doit offrir un système multimédia et celles qui sont plus spécifiques à la définition d'un SGBD multimédia.

2.1 Les données multimédias

2.1.1 Les types de données monomédias

Il existe un certain nombre de types de données caractérisées comme des données monomédias [BT93, GBT94, GDT94, KB96]. Dans ce paragraphe, nous décrirons brièvement les différents types de données qui caractérisent un système multimédia :

- **le texte**

  Le texte est en effet considéré comme une donnée multimédia. La donnée *Texte* est souvent réduite à une représentation d'une liste de chaînes de caractères. Pourtant une meilleure représentation de l'information textuelle, telle que les archives des documents multimédias, devrait inclure de l'information structurelle comme le titre, les auteurs, l'affiliation des auteurs, le résumé, les sections, les sous-sections et les paragraphes. En plus de cette représentation de la structure logique du texte, une représentation descriptive de l'information textuelle représente de l'information sur la mise en page [NXN91, ISO92]. Ces représentations sont utilisées par le processus de recherche. Par exemple, des opérateurs de recherche peuvent être définis sur la structure logique et sur la structure de mise en page. Ceci nécessite une extension des langages de requêtes offerts par les systèmes de bases de données et une intégration des concepts connus de la

recherche d'informations.

- **l'image**

  Le type *Image* (ou *Picture*) correspond à des dessins, de la peinture, de la photographie ou de l'impression. Un système multimédia, comme un dictionnaire électronique d'art ou une encyclopédie qui présente toutes sortes de peintures ou photographies, doit ainsi posséder des fonctionnalités pour importer et manipuler ces médias. Les opérations de manipulation de base sont, par exemple, le *découpage*, la *correction chromatique* et la *composition de différentes sources d'images*. Manipuler ce type de données pour un SGBD nécessite de pouvoir gérer des données structurées de taille importante, mais aussi pouvoir les interroger sur leur contenu [ABF+95, OS95, FSNA95].

- **l'audio**

  Par rapport aux deux précédents types qui partagent la caractéristique d'être indépendants du temps, l'*Audio* dépend du temps. Une façon d'interpréter la donnée audio est basée sur sa relation avec une échelle de temps progressivement constante. L'échelle de temps associe à la donnée audio, ou plus précisemment aux constituants atomiques d'un flux audio, une interprétation correcte à chaque point du temps [GBT93, LR94]. Certaines opérations de manipulation comme *couper*, *copier* ou *coller* peuvent être effectuées statiquement. D'autres opérations comme *jouer* et *enregistrer* seront toujours associées avec une échelle de temps. Dans le cas de l'audio, l'échelle de temps est absolue et correspond au temps du monde réel. Enfin, la donnée audio représente une masse de données importante qui nécessite des techniques de compression pour son stockage ou son échange entre composants du système.

- **la vidéo**

  Le type *Video* intègre les propriétés de la donnée *Audio* et de la donnée *Image*. L'échelle de temps de la vidéo est absolue et associe à chaque image son interprétation correcte à chaque point du temps. Les opérations de manipulation comme *couper*, *copier*, *coller*, *jouer* et *enregistrer* sont similaires à celles définies pour la donnée audio. Les composants atomiques de la vidéo sont les images qui correspondent à des données de type *Image*. Pour permettre une recherche sur le contenu, les opérateurs de recherche peuvent être définis sur le contenu d'une vidéo, par exemple, rechercher les portions particulières d'une vidéo qui démarrent par des morceaux de scènes spécifiques qui se rapprochent d'une image donnée

[WDG94, HR96, LM97]. La vidéo possède aussi une structure hiérarchique formée de séquences, scènes et plans [GBT92, ACC+96, Loz97]. La représentation de la donnée vidéo nécessite aussi des techniques de compression [LG91a].

2.1.2 Volume des données

Une des caractéristiques des données multimédias est leur grand volume [BGMLS93, RNL95]. La table 2.1 illustre la taille des données pour différents types de médias. Ceci montre clairement que la capacité de stockage doit être entièrement définie en fonction du type de données, et des formats de ces données. Le rapport entre l'espace physique pour 500 pages de texte au format ASCII et celui pour 500 images (format GIF) est de 1 pour 1000.

Type de média	**Format**	**Volume de données**
Texte	ASCII	1 MB / 500 pages
Image noir et blanc		32 MB / 500 images
Image couleur	GIF, TIFF; JPEG	1.6 GB / 500 images 0.2 GB / 500 images
CD-audio	CD-DA	52.8 MB / 5 minutes
Vidéo	PAL	6.6 GB / 5 minutes
Vidéo de haute qualité	HDTV	33 GB / 5 minutes
Audio	m-law, linear; ADPCM, MPEG audio	2.4 MB / 5 minutes 0.6 MB, 0.2 MB / 5min.

TAB. 2.1 – *Les types de média, leur format et leur volume de données relatif.*

Ces gros volumes de données posent le problème du stockage. Des méthodes de gestion, de distribution et de transfert de ces données tenant compte de leurs spécificités doivent être développées en relation avec les techniques de stockage. Par exemple, l'utilisation de procédés de stockage optiques semble nécessaire malgré les performances de temps d'accès de ces structures de stockage. En effet, le but de ces systèmes est de pouvoir traiter les données en temps réel, et ceci passe par l'utilisation de structure de stockage permettant un accès immédiat aux données (disques magnétiques parallèles). Le stockage physique des données dépend aussi de l'organisation logique adoptée par le système (fichiers, bases de données relationnelles ou objets). Cette tâche paraît essentielle dans le développement d'un système multimédia, car l'accès aux données est fortement lié à la structure et aux outils de gestion de ces données.

La compression des données est le principal moyen pour réduire la taille pour le stockage et la transmission de ces données. Différentes techniques de compression sont connues, mais le but général est d'avoir une technique rapide, se faisant en une unique passe, adaptative, réversible et enfin qui puisse être effectuée avec des moyens raisonnables. Dans ce domaine, les techniques de compression des données de type texte se distinguent des techniques de compression des images, de la vidéo ou du son. En effet, la compression des données part du postulat que tous les bits d'information sont importants, alors que la compression des données visuelles ou sonores considèrent que seule la partie "pertinente" de l'information doit être conservée. Les ratios sont donc différents.

2.1.3 Représentation des données

Tout d'abord, il est nécessaire de construire de nouveaux types de données et de nouvelles opérations pour les données multimédias. En effet, les types de données de base comme les alphanumériques ne sont pas appropriés pour la structure des données multimédias. De même, il est nécessaire de rajouter des opérations telles que l'édition de vidéos, la synchronisation de l'audio et de la vidéo, la fonction play.

La représentation des données multimédias nécessitent une représentation modulaire et efficace de différents formats. Les données multimédias utilisent différents formats pour un type de données à cause:

- de l'existence de différentes techniques de compression.

- la représentation interne peut ne pas être appropriée pour la présentation à l'utilisateur. Or cette représentation des données doit être transparente à l'application et à l'utilisateur.

2.2 Les systèmes auteurs

Différentes approches technologiques ont été développées pour réaliser des systèmes capables de développer des applications multimédias. Les systèmes auteurs en font partie et sont parmi les plus répandus. Un système auteur est une interface utilisateur qui permet à un auteur de décrire des *scénarios temporels* pour des applications

multimédias. Ils possèdent les caractéristiques suivantes :

- ils facilitent le développement d'une application multimédia (accessible à un non-informaticien) en produisant un environnement unifié pour leur création. Un système auteur est un programme qui possède des éléments préprogrammés pour le développement de logiciels multimédias interactifs.
- un système auteur typique inclut de l'hypertexte, des images, des animations, des vidéos, des sons, une connexion à une base de données et une communication entre processus. La connexion à la base de données permet seulement le stockage des données multimédias brutes. Elle n'offre aucune autre fonctionnalité; par exemple, les scénarios multimédias créés à l'aide du système ne sont pas stockés dans la base, aucune recherche n'est envisageable.

Les systèmes auteurs varient dans leur orientation, leurs capacités et leur philosophie. Un système auteur est à l'heure actuelle juste une forme plus rapide de programmer. Il n'est pas nécessaire de connaître la complexité d'un langage de programmation, mais seulement de comprendre comment les programmes tournent (leur utilisation nécessite quelques connaissances d'algorithmie). Fondamentalement, les systèmes auteurs se situent dans l'une des catégories suivantes :

- *Structure imbriquée*

  L'information temporelle est imbriquée dans la structure de composition des objets multimédias. Par exemple, on peut utiliser la structure de graphes pour exprimer la communication et l'ordre de présentation ou encore utiliser la structure hiérarchique d'un arbre pour spécifier les relations parallèles et séquentielles.

- *Graphisme*

  Les sytèmes auteurs graphiques produisent un outil avec une interface utilisateur visuelle pour spécifier l'information temporelle. Par exemple, le système auteur multimédia CMIFed [vRa93] produit une visualisation à l'aide d'une ligne de temps traditionnelle et une visualisation de type hiérarchique dans des fenêtres d'emboîtements.

- *Langage de script*

  De nombreux systèmes multimédias adoptent le langage de script parce qu'il s'agit d'une méthode puissante pour définir les scénarios temporels. Un langage de script orienté-objet performant est actuellement la pièce principale d'un tel

système. Le langage produit plus d'informations sur la présentation et permet de réaliser une interactivité beaucoup plus puissante. L'aspect négatif de ce type de langage est la difficulté à les apprendre et les utiliser, l'utilisation du script tend à augmenter le temps de développement. Comme la complexité du contrôle de la cohérence dans un scénario augmente fortement lorsque le script grossit, le langage de script n'est pas vraiment adapté au développement d'un gros système.

Une étude a été réalisée sur trois systèmes auteurs du marché [CC96d]. Chaque système utilise une métaphore différente pour donner une vue conceptuelle d'une présentation : un *livre* pour `Toolbook`, des *icones* pour `IconAuthor` ou une *vidéo* pour `Director`.

Multimédia Toolbook 3.0 - Ce système utilise la métaphore du livre. Un livre est construit en créant des pages, en plaçant des objets sur les pages et en écrivant des scripts pour exécuter des actions dans le langage "Toolbook's Openscript". On ne parle plus d'écrans pour l'application mais de pages. Toolbook a un environnement de programmation basé sur des événements orienté-objet, cependant le langage de script n'est pas orienté-objet (il n'utilise pas de classes, d'héritage ou d'encapsulation). Une page pour la définition d'une requête est construite à partir de boutons ou de menus déroulants et est générée sous forme SQL puis envoyée à la Base de Données via ODBC (Open Database Connectivity). Enfin, si l'utilisateur clique sur une vidéo résultante, le système lance "Quicktime Player" et joue le fichier MPEG associé. D'après leur étude, Toolbook est finalement un outil puissant pour les applications multimédias auteurs. Malheureusement, même une simple application nécessite une partie de programmation signifiante.

IconAuthor 6.0 - Les applications sont décrites en utilisant un organigramme d'icones. Un icone est une petite image qui représente une fonction à exécuter. Une application commence à l'icone "Start" et exécute l'icone qui le suit immédiatement et de même pour les icones suivants. Il est important de noter que le flot de contrôle est procédural et non basé sur les événements.

Director 4.0 - Ce système utilise une métaphore "vidéo" pour créer des applications. `Director` utilise un langage de script orienté-objet. Au cours de l'implantation de l'application, certaines déficiences importantes sont apparues comme le manque d'une liaison à une base de données ou l'impossibilité d'importer un fichier texte au format RTF (ou autre).

En conclusion, les systèmes auteurs multimédias sont attractifs pour des utilisateurs naïfs ou experts. Ils permettent à un utilisateur de décrire comment différents

objets peuvent être composés ensemble pour créer de nouveaux types d'objets multimédias. Mais leurs approches sont limitées, ils considèrent seulement les données non conventionnelles pour construire des présentations multimédias et ne considèrent pas les données alphanumériques comme de l'information traditionnelle comme le font les SGBD [ALMM97]. Ils ne prennent pas en compte les liens sémantiques entre les données multimédias et les données traditionnelles (par exemple les attributs descriptifs d'un objet `Person` et sa photo qui peuvent apparaître dans une présentation). Cette séparation implique certaines limitations, par exemple, au niveau du processus de recherche d'informations. Il est impossible d'accéder aux données multimédias stockées dans la base.

La grande faiblesse des systèmes auteurs se situe donc au niveau de l'interrogation des scénarios temporels. Certains systèmes sont reliés à des bases de données mais seules des données multimédias brutes sont stockées. Elles ne stockent pas les scénarios avec leur structure, leurs aspects temporels ou de synchronisation. Il est ainsi impossible de retrouver des scénarios existants en effectuant des recherches basées sur ces caractéristiques. Cette faiblesse entraîne le fait que les systèmes auteurs ne sont pas capables de proposer la notion de réutilisabilité des scénarios. Nous montrerons, au cours de cette thèse, que la technologie des SGBD permet de proposer des solutions pour l'interrogation et la réutilisabilité des scénarios multimédias.

De plus, les systèmes auteurs ne sont pas nécessairement cohérents avec le modèle de synchronisation dans un système multimédia. Par exemple, un scénario temporel peut être décrit dans un langage de script mais représenté par un graphe de flux de temps. Lorsqu'un scénario d'un système auteur est traduit dans le modèle de synchronisation, la question consiste à savoir si aucune information n'est perdue lors de ce processus.

2.3 Les systèmes multimédias

Dans un système multimédia, les données ajoutées, par rapport aux systèmes traditionnels, sont la vidéo, l'audio, les images, les graphiques, le texte. La principale caractéristique d'un **système multimédia** est donc l'intégration des données multimédias avec des données de type conventionnel (entier, chaîne de caractères ...), mais aussi l'inclusion de leurs opérations et de leurs contraintes [Chr94, HSR94, Gha95, CMCSCL95, KB96].

Un **SGBD multimédia** est un *système multimédia* qui produit une gestion de

données appropriée aux données multimédias et les services d'un SGBD classique [WK87, KA95, NTB96, ABH97].

Nous allons décrire maintenant, à partir de différentes approches, les différentes caractéristiques des systèmes multimédias qui peuvent être utilisées pour le développement des futurs SGBD.

2.3.1 Les aspects temporels

Au niveau de la dynamique d'un document multimédia, on distingue deux aspects: premièrement, il existe les aspects dynamiques inhérents aux types de données multimédias audio et vidéo; deuxièmement, la dynamique d'un document multimédia est aussi basée sur les relations temporelles entre les constituants de type média (présentation parallèle d'une vidéo et d'un son). Il est ainsi nécessaire d'intégrer les concepts temporels dans le modèle de données.

2.3.1.1 Les modèles temporels

Dans [WR94], on trouve une étude intéressante de différents concepts pour représenter le temps dans les systèmes multimédias. Deux grands modèles temporels caractérisent la plupart des approches courantes : le *modèle par instants* et le *modèle par intervalles*.

1. Dans un *modèle par instants*, comme dans HyTime [NXN91, VB95] ou [GBT93] ou les réseaux de points temporels [BZ93], l'unité temporelle de base est un instant de temps, c'est à dire un point sur la ligne de temps. Ainsi, les événements de début et de fin des objets multimédias sont associés à des instants. Ce type de modèle semble bien approprié aux compositions temporelles d'objets multimédias dont les instants de début et de fin sont clairement déterminés. Cependant, il force l'utilisateur à déterminer avec précision le début et la fin de chaque objet multimédia, ce qui n'est pas toujours simple dans des compositions temporelles complexes.

2. Dans un *modèle par intervalles*, le composant de base d'une relation temporelle est un intervalle de temps, correspondant à une durée non nulle dans le temps [AN86, Adi90, LG91b, LG93]. Elle est caractérisée par deux instants [All83]. Un instant dans le temps est un moment dans le temps avec une durée nulle, comme "4:00 PM". Par contre, un intervalle de temps est défini par deux instants du

temps et, ainsi, par sa durée ("100 ms" ou "huit heures" sont des intervalles temporels).

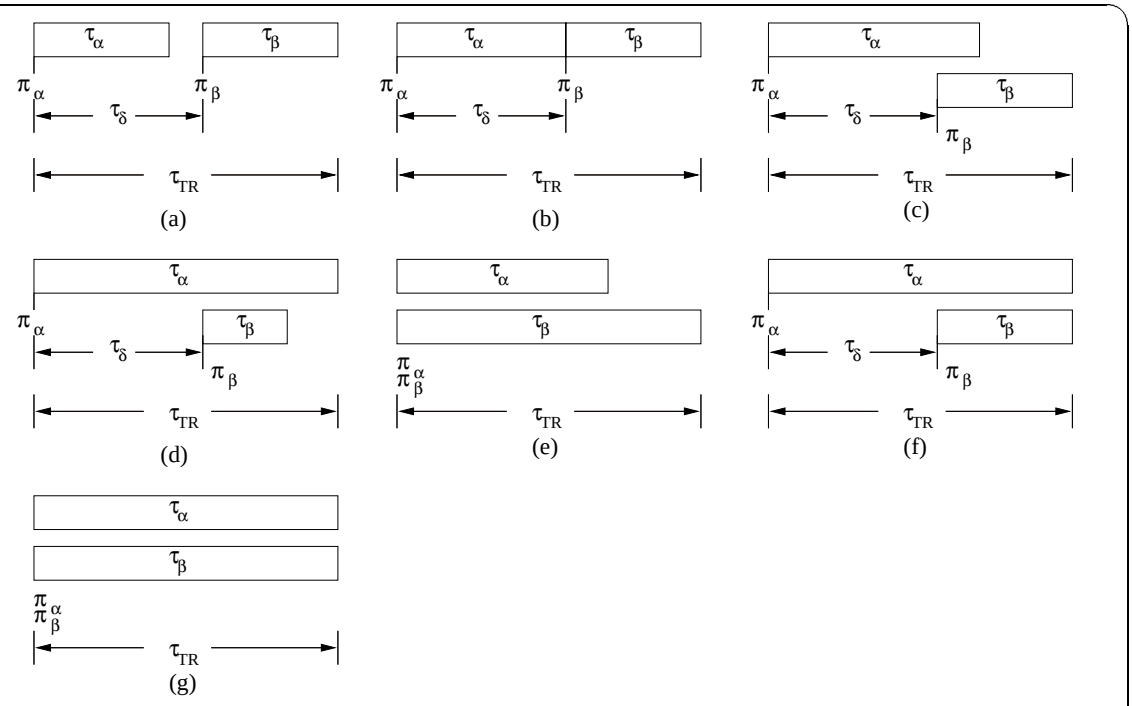

FIG. 2.1 – *Les relations temporelles binaires : (a) α before β, (b) α meets β, (c) α overlaps β, (d) α during^{-1} β, (e) α starts β, (f) α finishes^{-1} β, (g) α equals β.*

Pour deux intervalles donnés, il existe treize relations différentes possibles entre eux. Ces treize relations peuvent être représentées seulement par sept cas, puisque six d'entre elles sont en fait des relations inverses. La figure 2.1 présente graphiquement ces relations, en utilisant la représentation de [All83] et de la ligne de temps. Pour chaque élément de données α et β, les intervalles ont pour instants de début : π_α, π_β, et pour durées : τ_α, τ_β. La position relative entre eux est représentée par un délai τ_δ démarrant au début du premier intervalle et la durée totale est représentée par τ_{TR}.

Les intervalles temporels peuvent être utilisés pour modéliser une présentation multimédia, chaque intervalle représentant la présentation d'un élément de données multimédias, comme une image ou un son.

De nombreuses et intéressantes approches se basent sur le modèle par intervalles [CC97], parmi lesquelles :

- le *réseau de Petri composite à objets OCPN* (Object Composite Petri Net) [LG90, IDG94] : cette approche est basée sur les mêmes relations temporelles, mais elles sont représentées à l'aide d'un réseau de pétri comme le montre la figure 2.2.

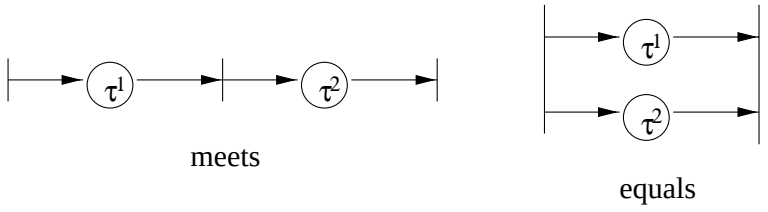

FIG. 2.2 – *Représentation des relations temporelles* `meets` *et* `equals` *à l'aide d'un réseau de pétri.*

OCPN est un modèle de réseau de pétri étendu. La "place" indique l'intervalle ou le délai associé à l'objet média, tandis que les "transitions" synchronisent les points de fin des intervalles. Le délai peut être soit une valeur fixe, soit un symbole inconnu comme le montre la figure 2.3. En d'autres termes, ce modèle supporte les objets médias sans connaître exactement leur temps de présentation à l'avance en insérant une place pour le délai à un point donné de la synchronisation. Par exemple, sur la figure 2.3, après avoir affiché l'objet `video1` et après deux unités de temps, le système démarre la présentation de l'objet `audio1`. De même, l'objet `text1` démarre après le début de `video1`, mais après une période inconnue.

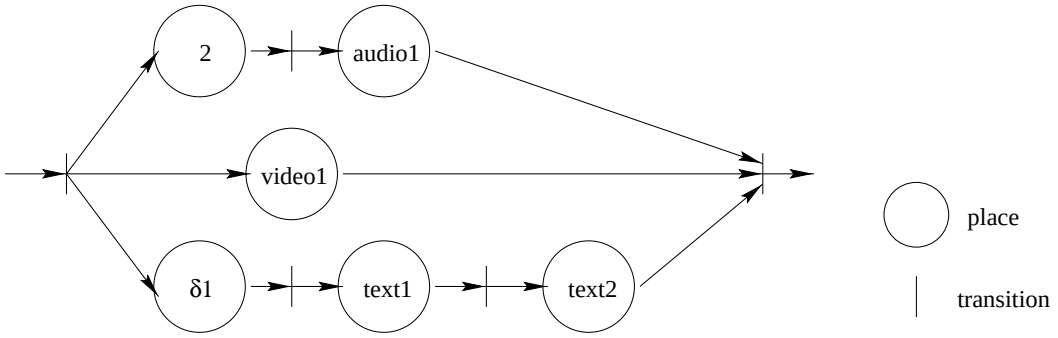

FIG. 2.3 – *Un exemple de réseau de pétri étendu en OCPN*

Les avantages d'un modèle de réseau de pétri est qu'il est adapté aux aspects de concurrence et de synchronisation, et qu'il existe une théorie bien établie sur ce modèle. Ainsi, il est possible d'analyser les caractéristiques et les comportements

du modèle. Cependant, il est difficile d'exprimer la synchronisation à la fin des objets avec un délai exact dans le temps.

- le *modèle de la ligne de temps* : cette approche est employée par HyTime [ISO92] et QuickTime [Com91]. Dans ce modèle, tous les objets sont alignés sur un axe de temps. Chaque intervalle d'objet multimédia est associé avec un axe et non pas avec les autres objets. La figure 2.4 montre un exemple de représentation temporelle pour une ligne de temps. Les avantages de la représentation à l'aide de la ligne de temps sont la facilité d'obtenir les points de synchronisation au niveau de la présentation, un support simple pour les mécanismes de synchronisation et la possibilité de modifier l'intervalle des objets sans affecter les autres. Et le fait que les objets médias sont alignés selon des lignes de temps indépendantes les unes des autres garantit une certaine cohérence. Cependant, il est difficile de supporter les objets médias sans connaître leur temps de présentation à l'avance.

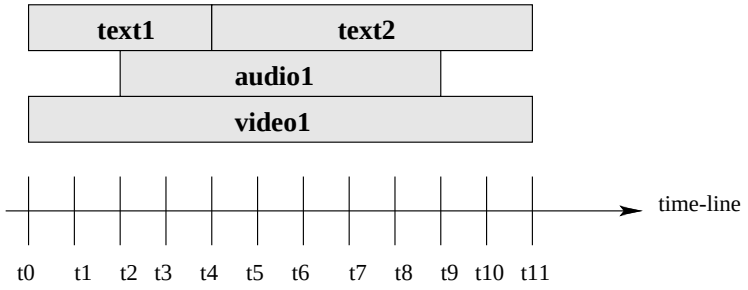

FIG. 2.4 – *Un exemple de représentation à l'aide de la ligne de temps*

- le *graphe de flux de temps*: aux premiers abords, certains graphes de flux de temps sont similaires aux modèles de réseaux de pétri sauf qu'ils utilisent des arcs et des cercles. Cependant, un des avantages du graphe est qu'il est facile d'introduire de nouveaux symboles représentant de nouveaux opérateurs ou de nouvelles fonctions dans le système de présentations multimédias [KEM97].

La figure 2.5 présente un exemple de graphe de flux de temps tiré de [LKG94]. Les objets médias `A1` et `A2` démarrent simultanément. Lorsque `A1` et `A2` se terminent, `P1` démarre. Mais lorsque `A1` ou `A2` se termine, `P2` démarre. Dans cet exemple, le rectangle `N1` représente "rencontrer l'intervalle qui se termine en dernier" alors que le triangle représente "rencontrer le premier intervalle qui se finit".

Firefly [BZ93] utilise un autre ensemble de symboles pour exprimer l'information temporelle comme le montre la figure 2.6. Le cercle représente un événement dans

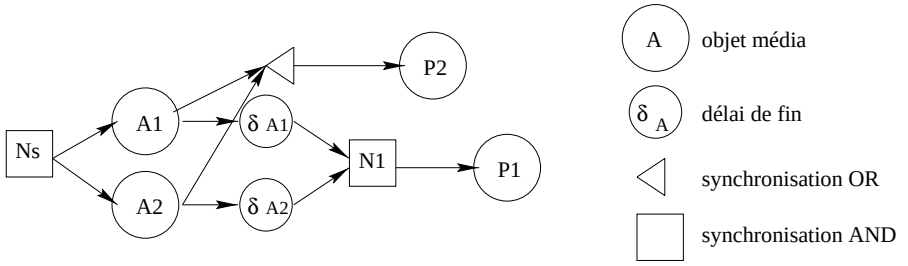

FIG. 2.5 – *Un premier exemple de modèle de graphe de flux de temps*

le système de présentations multimédias tel que le démarrage de la présentation d'un objet, et l'arc spécifie la synchronisation entre les événements.

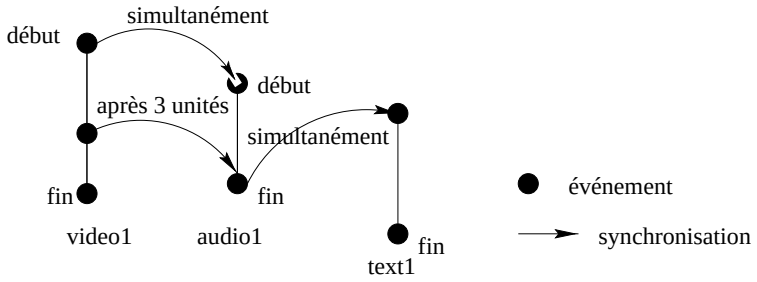

FIG. 2.6 – *Un deuxième exemple de modèle de graphe de flux de temps*

Les avantages du graphe de la ligne de temps sont la flexibilité et la précision dans la représentation de l'information temporelle en utilisant différents symboles. Cependant, l'utilisation de différents symboles est un inconvénient pour supporter les mécanismes de synchronisation puisqu'elle empêche la portabilité entre différents systèmes.

D'autres approches récentes, telles que celle de [DK95, KD96], sont proposées pour permettre une spécification temporelle basée sur des relations de causalité entre intervalles. Ils présentent un ensemble d'opérations basées sur des relations de causalité entre intervalles. Il existe six opérateurs de composition temporelle: `seq`, `follow`, `par_min`, `par_max`, `par_begin`, `par_end`, `equal`, `ident` présentés sur la figure 2.7.

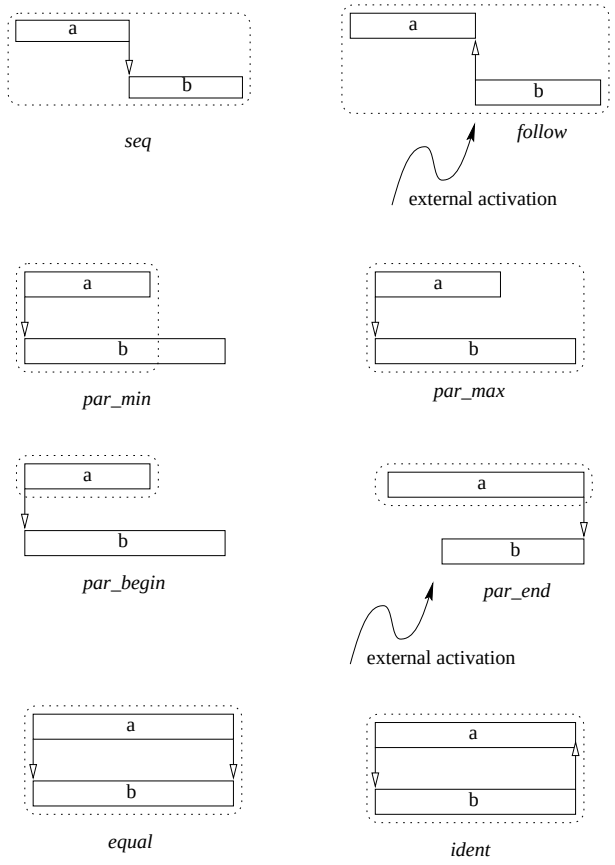

FIG. 2.7 – *Opérateurs de composition temporelle.*

Donnons pour chaque intervalle sa sémantique et la définition de l'intervalle résultat :

- **seq**(a, b) définit que la fin de l'intervalle a démarre l'intervalle b.
- **follow**(a, b) définit que le démarrage de l'intervalle b stoppe l'intervalle a; l'intervalle b étant activé de l'extérieur.
- **par_begin**(a, b) définit que le démarrage de l'intervalle a démarre l'intervalle b.
- **par_end**(a, b) définit que la fin de l'intervalle a stoppe l'intervalle b; l'intervalle b étant activé de l'extérieur.
- **par_min**(a, b) définit que le démarrage de l'intervalle a démarre l'intervalle b; l'intervalle résultat est stoppé quand le premier des deux intervalles se termine.

- **par_max**(a, b) définit que le démarrage de l'intervalle a démarre l'intervalle b; l'intervalle résultat est stoppé quand le dernier des deux intervalles se termine.
- **equal**(a, b) définit que l'intervalle a démarre et stoppe l'intervalle b.
- **ident**(a, b) définit que le démarrage de l'intervalle a démarre l'intervalle b et la fin de l'intervalle b stoppe l'intervalle a.

La définition de relations de causalité entre intervalles permet de ne pas connaître précisément la durée de ces intervalles, elle peut être indéterminée. Le démarrage ou la fin d'un intervalle sont toujours relatifs au démarrage ou la fin d'un autre intervalle. Ainsi ces relations offrent une meilleure solution dans la définition de relations temporelles entre intervalles à durée indéterminée.

En plus des concepts exprimant les relations temporelles entre les constituants dynamiques médias, il est nécessaire de produire des mécanismes pour la synchronisation. Il faut pouvoir modéliser, stocker et traiter les relations temporelles entre les composants médias d'un document multimédia ou d'une présentation.

2.3.1.2 Vérification de la cohérence temporelle

Les travaux de [LK95, LSI96b, Lay96] proposent des solutions pour le problème de la cohérence des documents multimédias. Un document multimédia est, en général, élaboré de manière incrémentale. L'auteur le construit à partir de composants de base qu'il relie progressivement en spécifiant des relations d'ordre spatial, sémantique, logique ou temporel. Il se peut toutefois que l'adjonction de nouvelles relations introduise dans le document des incohérences en violant certains invariants de structure que doivent vérifier tous les documents.

Par exemple, si trois objets a, b et c présentés sur la figure 2.8-a vérifient les deux relations temporelles {a meets b} et {b meets c}, alors la spécification de la relation temporelle {c overlaps a} devient incohérente. Il s'agit d'une *incohérence qualitative*, elle résulte en effet de la nature des relations indépendamment des durées des objets qu'elles relient. Cependant, les spécifications de cohérence qualitative peuvent être *incohérentes quantitativement* par respect des durées des objets. Sur la figure 2.8-b, les spécifications {a meets b} et {a starts c} entraînent une incohérence quantitative de la spécification cohérente qualitativement {c overlaps b} si les durées de a et c sont respectivement de 20 et 30 secondes. Enfin, la figure 2.8-c présente le même exemple que précédemment mais l'intervalle c est ici un objet indéterminé dont la

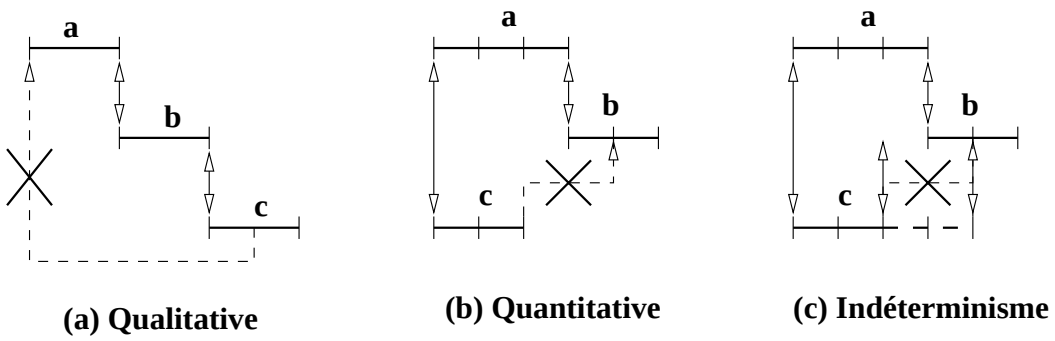

FIG. 2.8 – *Exemples d'incohérence temporelle*

durée est l'intervalle [20, 40], apparaît alors une incohérence due à l'*indéterminisme* puisque la fin de c peut se produire à chaque instant, ainsi cette spécification peut s'avérer fausse.

Ils proposent une manière de résoudre à la fois le problème de la maintenance de la cohérence temporelle en présence d'objets média indéterministes et l'efficacité de leur algorithme permettant ce genre d'opérations.

De la représentation abstraite d'un document, ils extraient un niveau supérieur de représentation de la structure temporelle pour modéliser les contraintes. Cette représentation définit un document multimédia comme un graphe de type DAG (Directed Acyclic Graph) où les noeuds sont les dates de "début-fin" des objets multimédias et les arcs sont des contraintes temporelles sur les durées. Ces contraintes reflètent les durées permises pour les intervalles et les délais spécifiés comme paramètres de relations temporelles. Pour vérifier la cohérence, ils établissent les deux états suivants :

- Les circuits ne sont pas permis puisque "les instants venant du passé ne peuvent pas être égaux aux instants dans le futur" et vice versa.
- Les cycles sont toujours composés de deux chaînes parallèles (propriété d'un DAG) dont les durées sont compatibles. La compatibilité temporelle de deux chaînes signifie que l'intersection des écarts des durées permises de deux chaînes n'est pas vide. Cette propriété impose que la cohérence temporelle des deux points de fin de deux chemins est garantie à l'exécution.

Ces deux états distinguent les sources des trois types d'incohérence introduits précédemment. Leur principale idée est de produire à l'aide de méthodes statiques les moyens de décider si un scénario donné peut être ajusté ou non dynamiquement. La

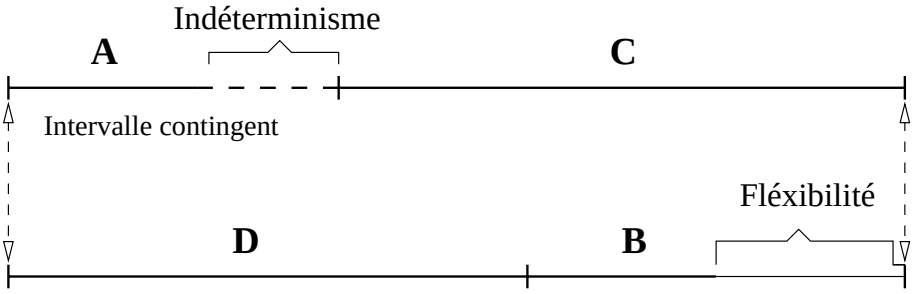

FIG. 2.9 – *Exemples d'incohérence temporelle*

figure 2.9 nous montre un exemple de réajustement d'un scénario. Ce scénario est constitué de deux chaînes qui doivent avoir des durées égales. Pour satisfaire cette contrainte, il faut imposer que les points de début et de fin de ces chaînes commencent et finissent en même temps. L'objet `A` est un objet "indéterminé" dont la fin peut se produire à chaque instant dans l'intervalle en pointillé. L'objet `B` est un objet continu avec une certaine fléxibilité représentée par une ligne plus fine sur la figure. Il est alors facile de constater que si la durée de fléxibilité de l'objet `B` est plus grande que l'intervalle d'imprévisibilité de l'objet `A` (condition 1) et que si le point observable précède le point de début de l'objet `B` (condition 2), alors la cohérence temporelle entre les instants de fin des objets `C` et `B` peut être vérifiée.

Leur proposition est intéressante pour une vérification temporelle de scénarios multimédias avec des présentations d'objets multimédias dont les durées sont indéterminées. Il faudrait aussi proposer un mécanisme proposant des solutions sous forme d'alternatives en cas d'incohérence temporelle dans un scénario, ainsi l'utilisateur ne redéfinit pas tout le scénario.

2.3.2 Les aspects spatiaux

La synchronisation temporelle est le processus qui assure que chaque élément apparaît à un temps voulu et joue pendant une certaine période. La composition spatiale décrit le processus de rassemblement d'objets multimédias sur un écran à un certain moment du temps. Le texte, les graphiques, l'image, et la vidéo sont appelés des données affichables.

Les travaux sur les aspects spatiaux sont en nombre plus limité. La définition des aspects temporels et de synchronisation est plus prioritaire. Il faut tout d'abord

définir comment se présentent les données au cours du temps et ensuite dans l'espace. Lorsque l'on crée une présentation, l'ordre le plus logique pour sa construction est de tout d'abord associer des aspects temporels aux objets qui la composent, ensuite définir les contraintes de synchronisation entre ces objets et enfin décrire les contraintes spatiales pour la présentation à l'écran des objets.

Nous verrons aussi, au cours de cette thèse, que notre attention s'est plus portée sur les aspects temporels. Ainsi, notre étude des aspects spatiaux est moins approfondie, nous présentons seulement l'approche de [Vaz96] qui propose des idées intéressantes pour la composition spatiale et peut être un point de départ pour de futures reflexions sur l'intégration des aspects spatiaux.

L'approche de [Vaz96] s'intéresse aux contraintes spatiales à l'aide de différents opérateurs binaires : {`disjoint`, `meet`, `overlap`, `covers`, `inside`, `equal`} présentés sur la figure 2.10.

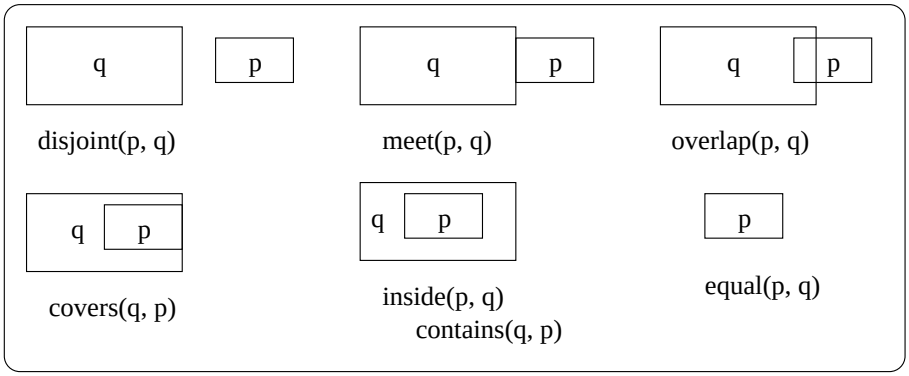

FIG. 2.10 – *Contraintes spatiales*

La sémantique de ces différents opérateurs est la suivante :

- **disjoint(p, q)** impose que les présentations des objets `p` et `q` ne se chevauchent pas à l'écran.
- **inside(p, q)** exprime que la présentation de l'objet `q` doit inclure la présentation de `p`, l'inverse est `contains(q, p)`.
- **equal(p, q)** impose que les présentations des objets `p` et `q` coincident.
- **overlap(p, q)** impose que les présentations des objets `p` et `q` se chevauchent à l'écran.
- **covers(p, q)** exprime que la présentation de l'objet `q` doit inclure la présentation de `p` et les deux présentations doivent se toucher.

- **meet(p, q)** impose que les présentations des objets `p` et `q` ne se chevauchent pas à l'écran, mais se touchent.

Ces différents opérateurs permettent de définir un certain nombre (déjà conséquent) de positions des objets à l'écran les uns par rapport aux autres, mais il faut penser aussi à définir les valeurs des distances entre les objets et vérifier que tout soit cohérent par rapport à l'affichage à l'écran (taille de l'écran, des fenêtres, etc.). Ainsi, la définition de contraintes spatiales va nécessiter des mécanismes de contrôles de la *cohérence spatiale*. Il faut vérifier, par exemple, que trois contraintes spatiales définies sur trois objets affichés au même moment peuvent se vérifier en même temps (ce qui n'est pas le cas de `inside(p, q)`, `disjoint (q, r)` et `meet(p, r)`).

2.3.3 Modélisation de l'information multimédia

La modélisation est une nécessité dans le domaine des bases de données. Pour pouvoir manipuler l'information multimédia, il faut avant tout la modéliser selon deux problématiques : (1) modéliser la sémantique des données multimédias pour permettre des recherches sur le contenu et (2) modéliser à la fois la composition des objets et la synchronisation qui exprime des relations temporelles et spatiales entre les objets.

2.3.3.1 Modélisation de la sémantique des données

La *sémantique des données* doit être modélisée pour permettre des recherches sur le contenu, ainsi que sur l'information textuelle associés aux données multimédias. Il est nécessaire de fournir des mécanismes d'indexation pour ces données, il faut pouvoir leur associer une information pertinente dans le but de les interroger. Le processus d'indexation peut être automatique ou manuel.

Les travaux de [AC95, ABH97] apportent une solution intéressante à la modélisation de la sémantique des différents types de média, permettant ainsi une recherche des données basée sur le contenu des objets. Ils proposent un modèle de représentation Orienté-Objet multimédia et un descriptif de la recherche basée sur le contenu dans les systèmes de Bases de Données texte, audio, image et vidéo.

Le *modèle de description multimédia* décrit la composition logique de l'objet multimédia, la synchronisation et les contraintes de temps entre ses composants, la position de la valeur média, et les paramètres pour l'affichage de la valeur média (voir figure 2.11).

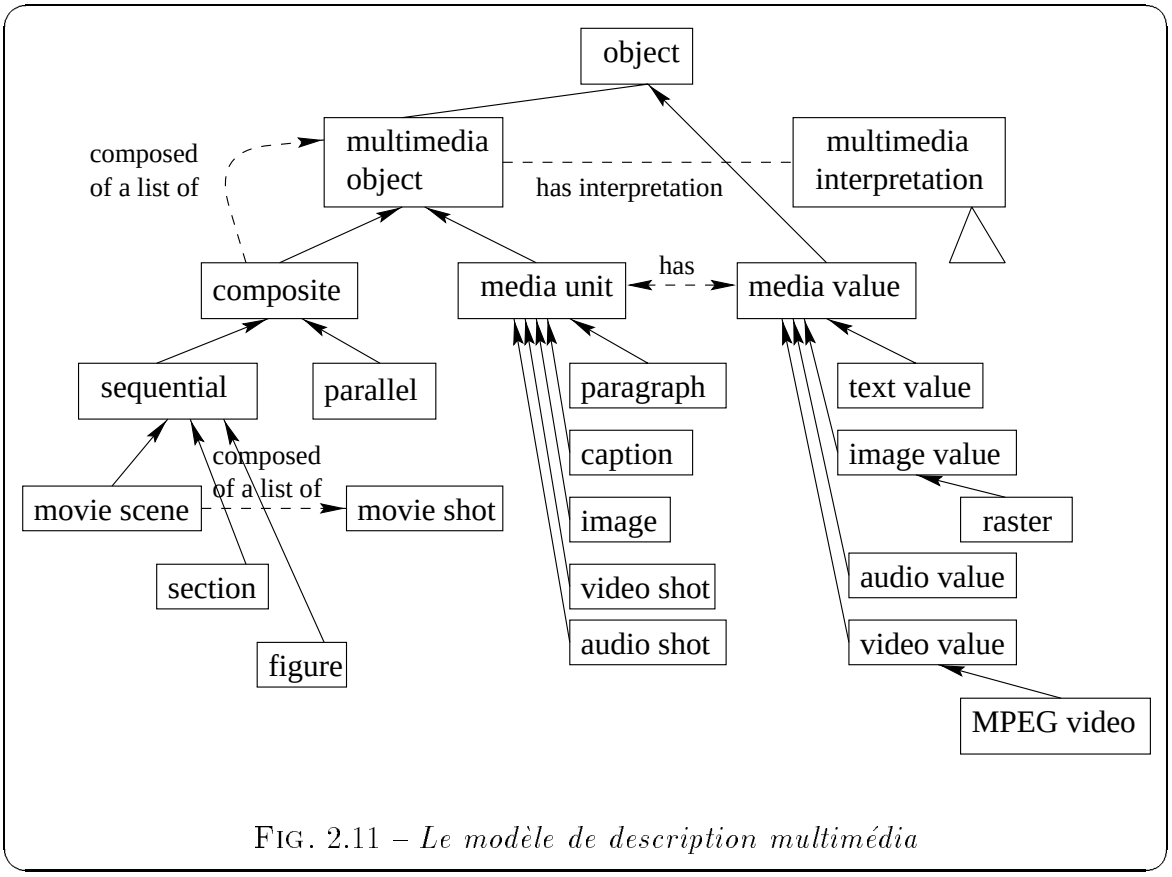

FIG. 2.11 – *Le modèle de description multimédia*

Chaque objet multimédia *composite* est composé d'une liste d'objets multimédias qui peuvent être soit simples, soit composés. Un objet multimédia simple est considéré comme l'unité de l'objet multimédia, c'est à dire du texte, de l'image, de la vidéo et de l'audio. Un objet multimédia composite peut être vu comme la racine d'un arbre dont les noeuds correspondent à des objets multimédias de type `sequential` ou `parallel` indiquant une composition séquentielle ou parallèle de ses enfants. Les feuilles de l'arbre sont des objets simples (`media unit`). Chaque objet simple a une valeur média (`media value`). Cet objet encapsule une séquence de bytes (un BLOB ou une partie d'un BLOB) et un descripteur. Le descripteur contient de l'information nécessaire à la présentation de la donnée (BLOB). Enfin, chaque objet multimédia contient un nombre d'attributs avec de l'information technique relative à l'objet dans son ensemble. Un document multimédia est un objet multimédia composé séquentiellement avec de l'information extra-bibliographique.

Le *modèle d'interprétation multimédia* interprète la valeur multimédia en décrivant les objets conceptuels contenus dans cette valeur et leurs relations (c.f. figure 2.12). Le modèle intègre l'interprétation de chaque unité présente dans le modèle de des-

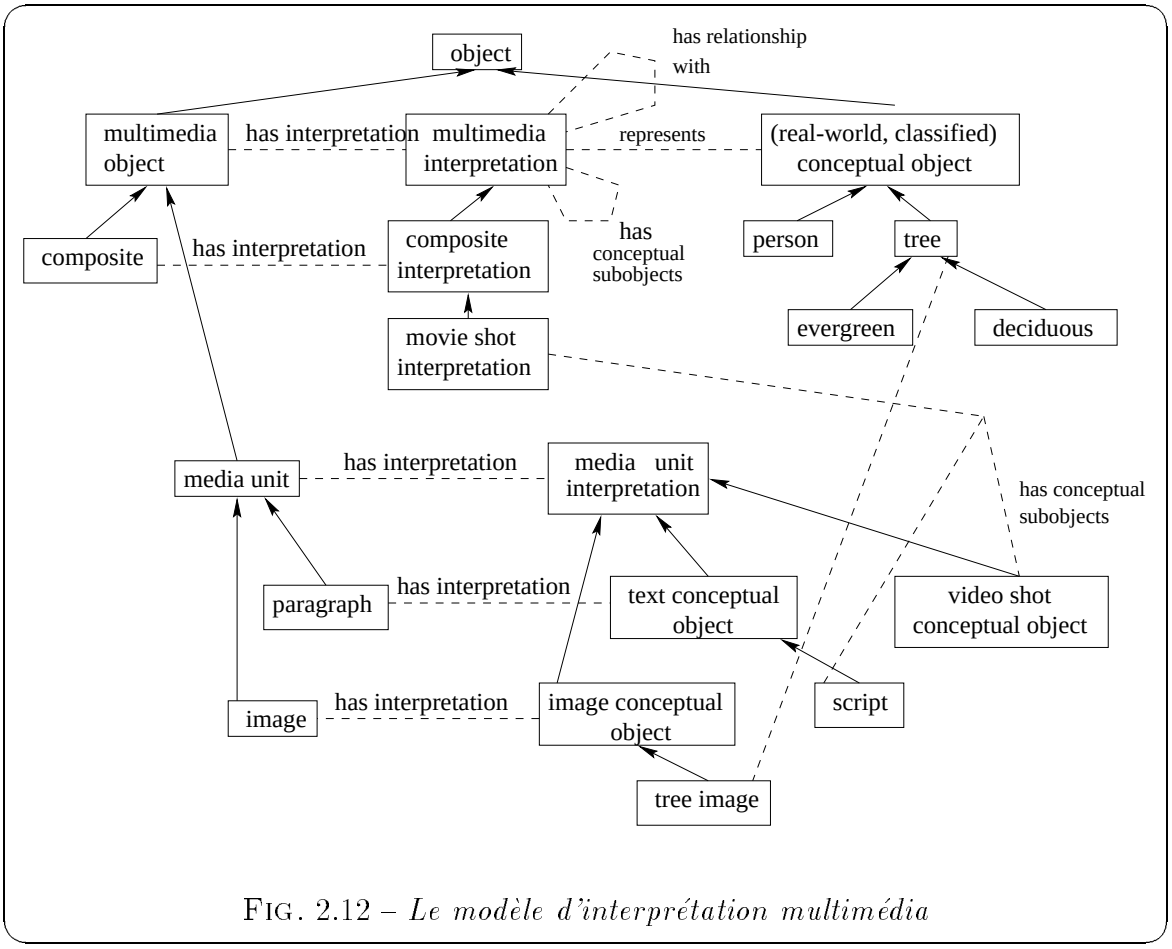

FIG. 2.12 – *Le modèle d'interprétation multimédia*

cription. L'interprétation du contenu d'un objet multimédia est stockée dans un objet `multimedia_interpretation` qui peut être un objet complexe. L'interprétation est définie à partir : (1) d'une hiérarchie d'objets représentant le monde réel et définissant les connaissances du domaine, il s'agit d'objets conceptuels classifiés. Par exemple, pour des images de visages, le domaine de connaissances k comprend : yeux, bouche, forme, etc. (2) des objets identifiés par leurs propriétés, mais qui n'appartiennent pas au domaine des connaissances. Ils sont définis comme des objets conceptuels non classifiés, leur existence dépend de l'objet multimédia où ils apparaissent. Par exemple, pour des images de visages, il s'agit des moustaches ou des lunettes.

D'autres approches proposent un modèle de données permettant de manipuler la structure des données en accord avec son type de média [GMY93]. Par exemple, la structure d'une vidéo est un ensemble de scènes. Chaque scène est elle-même composée d'une liste de plans et finalement chaque plan est composé par un certain nombre de coupures.

Il existe de nombreux travaux sur la modélisation de la donnée vidéo. Le modèle OVID [OT93] s'intéresse principalement à la manipulation d'annotations sur la vidéo. Le modèle utilise différents concepts comme la généralisation d'un graphe qui permet d'établir les relations entre les objets participant aux annotations de la vidéo. La définition des attributs avec des possibilités d'héritage optionnelles.

Dans le modèle "Track Algebra" [GR94, GRG95], le principal point est la trace qui permet différentes annotations sur les mêmes segments vidéo. Il existe différents types de traces comme les traces "média" qui représentent les différents médias ou les traces "annotation" qui annotent ces médias. Cette approche est intéressante pour la manipulation de flux de données de façon simple, mais les éléments (ou traces) ne sont pas structurés hiérarchiquement.

Le modèle AVIS [ACC+96] s'intéresse à la manipulation d'annotations de la donnée vidéo. Il propose des structures de données pour réaliser l'accès aux données. Il utilise des tables d'associations pour créer des index pour la donnée vidéo. Cependant, ce travail ne propose aucune opération pour la composition des données vidéo et aucune méthode pour spécifier la structure de la vidéo.

Le modèle VStorm [Loz96, Loz97, LM97] propose d'étendre un SGBD à objets en y ajoutant des capacités pour la gestion de la donnée vidéo. Pour cela, il répond aux problèmes suivants :

1. Le modèle propose une bibliothèque de classes qui définit la structure hiérarchique de la vidéo. Il considère qu'une vidéo est constitué de segments qui sont eux-mêmes composés de scènes qui peuvent être divisées en séquences. Il montre comment la structure hiérarchique d'une vidéo peut être modélisée et gérée par une approche objet.

2. Le modèle montre comment l'indexation du contenu de la vidéo peut être réalisée à l'aide d'une base de données à objets. Au lieu de se contraindre à une structure d'indexation, il propose un schéma générique (classes plus méthodes) qui peut être raffiné par l'utilisateur pour créer sa propre structure d'indexation.

3. Le modèle montre comment le SGBD peut être utilisé pour l'édition de nouvelles vidéos. A partir de vidéos stockées dans la base, une vidéo virtuelle est éditée et stockée comme une vue dans la base. Il offre ainsi la possibilité de créer de nouvelles vidéos avec les vidéos existantes dans la base sans dupliquer ces données.

4. Finalement, le modèle montre comment le langage de requêtes peut être étendu pour interroger les objets vidéo. La donnée vidéo peut être interrogée de la même façon que les données traditionnels et le modèle est capable de répondre au moins aux requêtes suivantes : (1) "Trouver toutes les séquences dans lesquelles apparaît Woody Allen", (2) "Trouver toutes les séquences où Woody Allen apparaît après Mia Farrows", (3) "Trouver toutes les séquences qui ont été filmées à Venise et dans lesquelles apparaît Woody Allen", et enfin (4) "Trouver toutes les séquences dans lesquelles apparaît Woody Allen et qui n'ont pas été filmées à New York".

2.3.3.2 Modélisation des présentations multimédias

Un modèle de données doit modéliser à la fois la composition des objets et la synchronisation qui exprime des relations temporelles et spatiales entre les objets [MM94, AK94, AL95, DHB96, Vaz96]. En effet, une présentation multimédia compose différents médias suivant une certaine synchronisation (parallèle ou séquentielle) dans le but de les présenter à un utilisateur. Les modèles de données conventionnels ne sont pas capables de répondre aux nouveaux besoins des applications, c'est pourquoi certaines extensions leur sont nécessaires pour concevoir de nouveaux modèles de données. Nous présentons deux approches différentes pour permettre de modéliser des présentations multimédias, mais utilisant toutes deux l'approche objet.

L'objectif du modèle proposé par [VH95, Vaz96] est la représentation d'objets multimédias et d'applications multimédias interactives. Ce modèle est spécifié comme une hiérarchie de classes (Figure 2.13).

Les objets multimédias sont classifiés en temporels et spatiaux comme le montre la figure 2.13. Ceci signifie que certains objets ont seulement des aspects temporels (comme pour les objets `audio`) ou les deux aspects (pour les images, les textes, les vidéos).

La classe `t_object` représente les caractéristiques temporelles des objets multimédias. Elles sont supportées par l'attribut `object_time` et les méthodes qui le manipulent. L'attribut `t_status` indique l'état de l'objet qui peut être soit actif, soit inactif, soit suspendu.

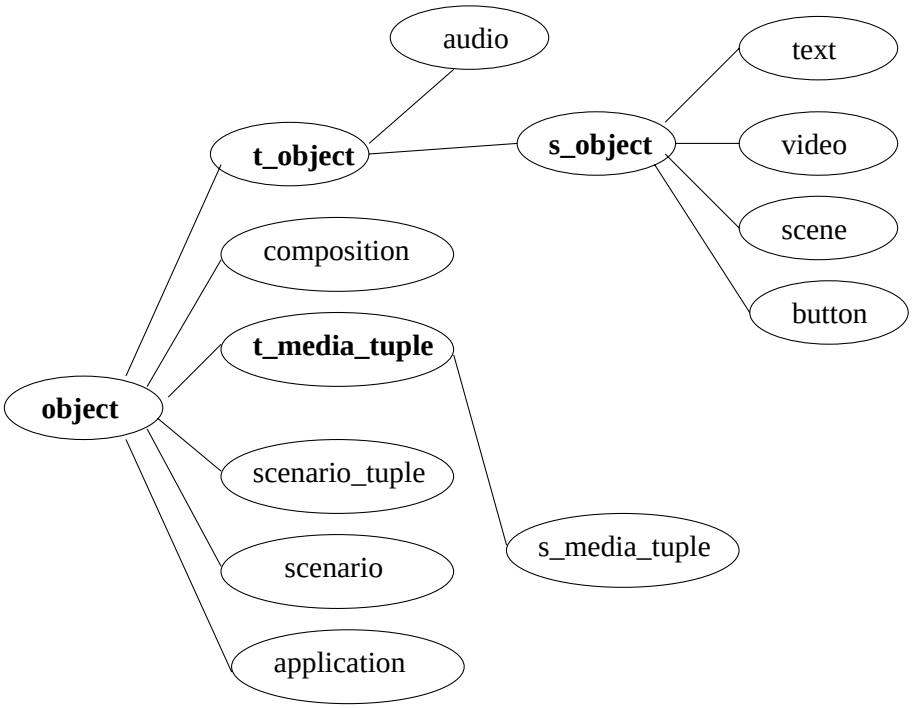

FIG. 2.13 – *La hiérarchie de classes proposée*

class t_object subclass of OBJECT
attributes

object_time	*domain TIME*
t_length	*domain TIME*
t_status	*domain INTEGERS*
t_scale_f	*domain FLOAT*
looping	*domain BOOLEAN*
direction	*domain BOOLEAN*

end

Les aspects spatiaux dans le modèle sont représentés par la classe `s_object` qui est une abstraction des caractéristiques et fonctionnalités des objets multimédias spatiaux (texte, image, vidéo). La classe `s_object` est une sous-classe de la classe `t_object` pour deux raisons :

- Les objets spatiaux peuvent aussi avoir en général des caractéristiques temporelles et des contraintes.
- La vidéo a des caractéristiques spatiale et temporelle.

class s_object subclass of t_object
attributes

m_width	*domain INTEGERS*
m_height	*domain INTEGERS*
plane	*domain INTEGERS*
transparency	*domain FLOAT*
focus	*domain BOOLEAN*

end

Les attributs de cette classe décrivent ses dimensions (`m_width` et `m_height`), et sa place dans l'application (`plane`). Ces méthodes manipulent la présentation de l'objet, permettent de bouger l'objet sur l'écran, mais aussi de définir la réponse de l'objet au système : `mouse_down`, `mouse_enter`, `mouse_leave`, etc.

Les relations temporelles sont définies à partir des relations d'Allen ([All83]) : `before`, `during`, `meets`, `overlaps`, `begins` et `ends`. Les relations spatiales sont les suivantes : `equal(p, q)` (p et q doivent coincider), `overlap(p, q)`, `meet(p, q)` (p et q doivent se toucher à l'extérieur), `covered_by(p, q)` (p et q doivent se toucher à l'intérieur), `inside(p, q)` et `disjoint`. Les aspects interactifs proposés dans ce modèle sont présentés au paragraphe 2.3.6.

La deuxième approche présentée se base sur la norme MHEG (Multimedia and Hypermedia information coding Expert Group) qui a été développée pour proposer un codage de l'information multimédia et hypermédia afin de pouvoir faire des échanges de ce type de données entre différentes applications [Pri93, Sa94b, MBE95]. Elle prend en compte l'aspect composite et hétérogène d'une donnée multimédia, l'aspect descriptif est important et permet une présentation directe prenant en compte le codage, la composition et le type de ressources nécessaires. Les applications visées sont l'éducation, les jeux, la publicité, la publication électronique, le médical, la télévision interactive, etc.

Cette norme se base sur une approche orienté-objet parce qu'elle répond aux besoins d'objets actifs, autonomes et réutilisables. La conception de classes d'objets relie leur comportement commun et leurs propriétés communes selon différentes catégories d'objets. Cette norme propose une hiérarchie de classes très complète permettant de spécifier les objets contenant une information multimédia, les associations entre ces objets, le comportement de ces objets et de l'information liée aux aspects temps-réel. Les instances des classes sont les objets échangés entre les applications. Les figures 2.14 et 2.15 décrivent respectivement les classes proposées par la norme MHEG et les

relations entre elles.

La norme définit aussi les "*run-time objects*" (ou `rt-object`): à partir d'une instance d'une classe MHEG, on peut créer n `rt-object` correspondant à n utilisations différentes. L'auteur peut ainsi définir de multiples vues pour le même objet.

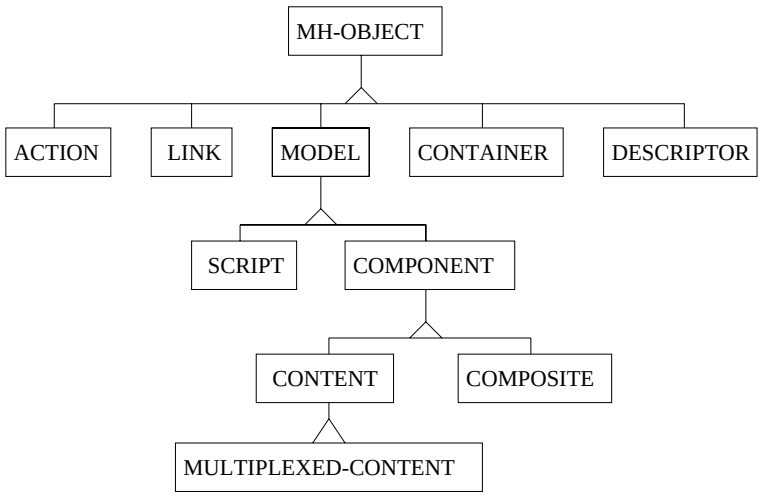

FIG. 2.14 – *Les classes MHEG*

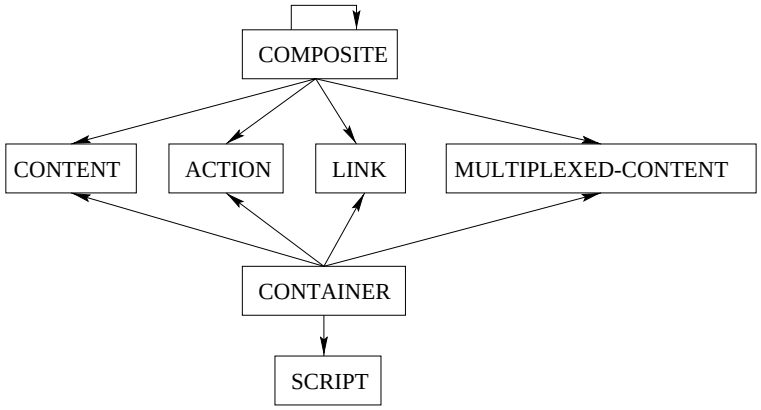

FIG. 2.15 – *Références entre les classes MHEG*

La classe `MH-OBJECT` est la racine de l'héritage et contient certains éléments relatifs à l'identification. La classe `ACTION` décrit un comportement pour chaque objet MHEG et chaque rt-object (modification du comportement initial). La classe `LINK` spécifie un ensemble d'associations entre une ou plusieurs sources et une ou plusieurs cibles (instances d'une classe MHEG quelconque). Une association est composée de conditions

liées à la source et d'actions à appliquer aux cibles. Les objets de la classe `LINK` sont utilisés pour définir des enchaînements dans le temps, des positionnements spatiaux ou des interactions entre objets MHEG et rt-objects. La classe `MODEL` est une classe abstraite ne contenant pas d'informations et qui sert à regrouper tous les rt-objects. La classe `SCRIPT` permet de définir des actions plus complexes (exprimées dans un langage particulier) entre des objets MHEG et des rt-objects. La classe `COMPONENT` est une classe abstraite ne contenant pas d'informations. Elle est le support général pour la manipulation de tous types de données encodées (multipléxées dans le cas de la classe `MULTIPLEXED-CONTENT`). La classe `COMPOSITE` décrit la synchronisation dans le temps et l'espace ainsi que les liens entre un ensemble d'objets. Un objet `COMPOSITE` référence des objets `ACTION` et `LINK` pour décrire le comportement dynamique et initial des objets composants et rt-objects. La classe `CONTAINER` permet de regrouper différents types d'information multimédia. Enfin, la classe `DESCRIPTOR` permet de décrire les besoins en ressource des différents objets MHEG. Cette description est destinée au moteur MHEG. Le moteur MHEG (ou *MHEG engine*) est responsable d'interpréter les objets MHEG pour reconstruire la structure de la présentation multimédia interactive.

Cette norme a un pouvoir d'expression puissant à l'aide des hiérarchies de classes définies, mais le problème est que la réalisation du moteur d'exécution est laissée au concepteur d'applications multimédias. En effet, la norme apporte des solutions aux besoins d'échange en définissant la représentation et le codage des objets contenant de l'information multimédia ou hypermédia, cependant, elle ne définit pas comment jouer à l'écran (le "look and feel") les présentations multimédias interactives.

2.3.4 Rechercher de l'information multimédia

Un système d'information multimédia doit proposer des recherches sur les aspects structurels, temporels ou sur le contenu des données multimédias [SLR94, FSNA95, OS95, Jag96]. La navigation classique n'est pas suffisante pour accéder aux données, il faut définir des langages de requêtes appropriés. La recherche de certaines données multimédias requiert l'utilisation de techniques d'indexation evoluées.

2.3.4.1 Recherche basée sur le contenu

Des systèmes ont été développés dans le but d'interroger des bases de données d'images, leur caractéristique étant la recherche basée sur le contenu à partir d'une indexation des images. QBIC (Query By Image Content) est un prototype de recherche

développé par IBM [FSNA95, ABF+95] qui est capable de retrouver des images en basant son mode d'interrogation sur leur contenu physique. Dans ce système, le contenu englobe, que ce soit pour l'image entière ou pour des objets (personnes, etc.) ou des régions physiques (ciel, mer, etc.), les couleurs, les textures et les formes. Ce système est une première avancée pour l'interrogation des données visuelles, en se concentrant principalement pour l'instant sur les images fixes, mais devant être étendu aux vidéos [CHN+95]. Les requêtes qui portent sur des propriétés physiques des images sont spécifiées graphiquement par l'intermédiaire d'une interface offrant la possibilité de dessiner un croquis, de sélectionner des couleurs, des textures, etc., pour chacune des régions ou objets du croquis. Plus généralement, ces outils permettent de spécifier les propriétés des images requises par l'utilisateur. Dans le même contexte de recherche, [OS95] est un système de recherche d'images pour une base de données. L'outil développé permet de répondre aux objectifs suivants : (1) développement d'une base de données relationnelle pour images et données diverses; (2) solutions de stockage; (3) proposer une navigation et une recherche en ligne dans la collection; (4) proposer un système de recherche simple et flexible; (5) offrir une recherche basée sur le contenu des images.

Les travaux de [AC95] décrits au début du paragraphe 2.3.3.1 (modélisation de la sémantique des données à l'aide d'un modèle de description multimédia et d'un modèle d'interprétation multimédia) permettent à l'utilisateur d'interroger les objets multimédias sur leur contenu en spécifiant : (1) les valeurs des attributs (date de création, auteurs, etc.); (2) les descriptions textuelles (les annotations); (3) les caractéristiques globales des objets multimédias (statistiques sur les couleurs, la texture...); (4) les propriétés ou relations spatiales des objets conceptuels; (5) le comportement temporel des objets conceptuels.

Les travaux de [BCF95] s'intéressent à un processus de requêtes dans un contexte multimédia. Un système multimédia supporte des objets avec une structure complexe, le principal problème est l'hétérogénéité de ces données. Cette variété de données influence le choix de la stratégie pour la meilleure exécution de requêtes. Il est nécessaire de définir des stratégies intégrées pour produire des requêtes multimédias, c'est à dire de combiner des stratégies définies pour chaque type de données de façon optimale. Ils définissent ainsi différentes dimensions dans un processus de requêtes, les composants actifs ou passifs, égalité partielle ou exacte. MULTOS est présenté comme un exemple de processeur de requêtes multimédias. Il permet l'archivage et la recherche de documents multimédias basés sur des collections de documents, des types de documents, des attributs de documents aussi bien que des documents de type texte ou image.

Les travaux sur la recherche des données multimédias basée sur le contenu sont nombreux dans le domaine de la recherche d'information. Les mécanismes développés sont encore des prototypes et ne sont pas encore totalement fiables dans le sens où ils rendent des résultats plus ou moins cohérents, une bonne partie du résultat n'a aucun rapport avec la requête énoncée. Mais ces systèmes deviennent de plus en plus puissants et ils seront facilement intégrables à un SGBD.

2.3.4.2 Interrogation des données multimédias

Produire des facilités de recherche sur les données multimédias est un problème crucial pour le domaine des bases de données multimédias. Les paradigmes conventionnels sur les requêtes des systèmes de bases de données traditionnels permettent seulement d'interroger avec exactitude les types de données conventionnels. Ils peuvent s'avérer suffisants pour pouvoir interroger les données multimédias sur leur abstraction, c'est à dire un certain nombre d'annotations décrivant la donnée. Ils sont plus limités lorsque l'on veut interroger les données multimédias sur leurs aspects temporels, spatiaux ou de synchronisation.

Les extensions des concepts conventionnels des langages de requêtes doivent prendre en compte les caractéristiques particulières des données multimédias. Les approches qui existent déjà et permettent de couvrir un grand nombre d'aspects sont les langages de requêtes temporels et spatiaux [HK96, HR96, Mar96]. Ces langages sont capables de traiter la sémantique temporelle et spatiale des données multimédias.

Les travaux de [HK96] proposent un langage de spécification de requêtes multimédias basé sur le langage MQL [KT94]. Dans le langage proposé, l'information multimédia est décrite selon la sémantique de son contenu. Le langage prend aussi en compte les spécifications spatio-temporelles aussi bien que les spécifications des médias contenus dans un document ou un segment de documents.

La syntaxe d'une requête est définie ainsi :
FIND X **IN** Doc-list
WHERE Multisegment-specification
où `X` est une variable référant les segments recherchés, `Doc-list` est une liste de documents sélectionnés dans la base, et `Multisegment-specification` est une description des segments qui nous intéressent. La description d'un segment se fait de la façon suivante :
Begin
le segment multimédia inclut des morceaux de vidéos sur le meeting du G7 en 1995,

de l'audio expliquant le rôle du Canada,
et du texte résumant les événements du G7.
End

Enfin, l'information multimédia perçue à l'aide des requêtes proposées peut être décomposée en trois différents types : (1) l'*Information temporelle* qui spécifie les relations dans le temps entre les différents objets multimédias; (2) l'*Information spatiale* qui spécifie les emplacements des objets multimédias dans l'espace; (3) l'*Information média* qui décrit les éléments individuels médias composant le segment.

Une autre approche, celle de [HR96], propose un langage de requête temporel visuel (TVQL) pour spécifier des recherches sur des relations temporelles entre des ensembles d'annotations. Le but de leur recherche est d'exploiter la *continuité* temporelle et les caractéristiques *spatio-temporelles* de la donnée vidéo pour pouvoir analyser la vidéo. Ils intègrent un paradigme de recherche visuelle à une présentation dynamique pour obtenir un environnement de visualisation interactif et convivial. Tout d'abord, leur approche permet d'identifier le contenu de la donnée vidéo dans le but d'interroger des relations entre des annotations de la vidéo. Ensuite, elle permet aux utilisateurs d'analyser la vidéo en termes de relations temporelles entre les événements.

Ces travaux montrent que le problème de l'interrogation des données multimédias est crucial. La technologie des bases de données est une des clés pour pouvoir stocker et interroger une grande quantité de données. Les langages de requêtes évoluent et permettent d'apporter des solutions aux différentes applications.

2.3.5 Une architecture de référence pour les systèmes de gestion de base de données multimédias

L'architecture des SGBD doit évoluer pour pouvoir intégrer de nouvelles technologies capables de manipuler, véhiculer, stocker ou interroger les données multimédias. La figure 2.16 nous présente une architecture de référence client/serveur proposée par [ABH97].

Au niveau du client, les modules nécessaires sont les suivants :

- le *gestionnaire de présentations* pour contrôler et gérer la présentation interactive de l'information multimédia.

- des périphériques de présentations spécifiques à un média qui peuvent être combinés à des périphériques de présentations virtuels pour la présentation de l'information

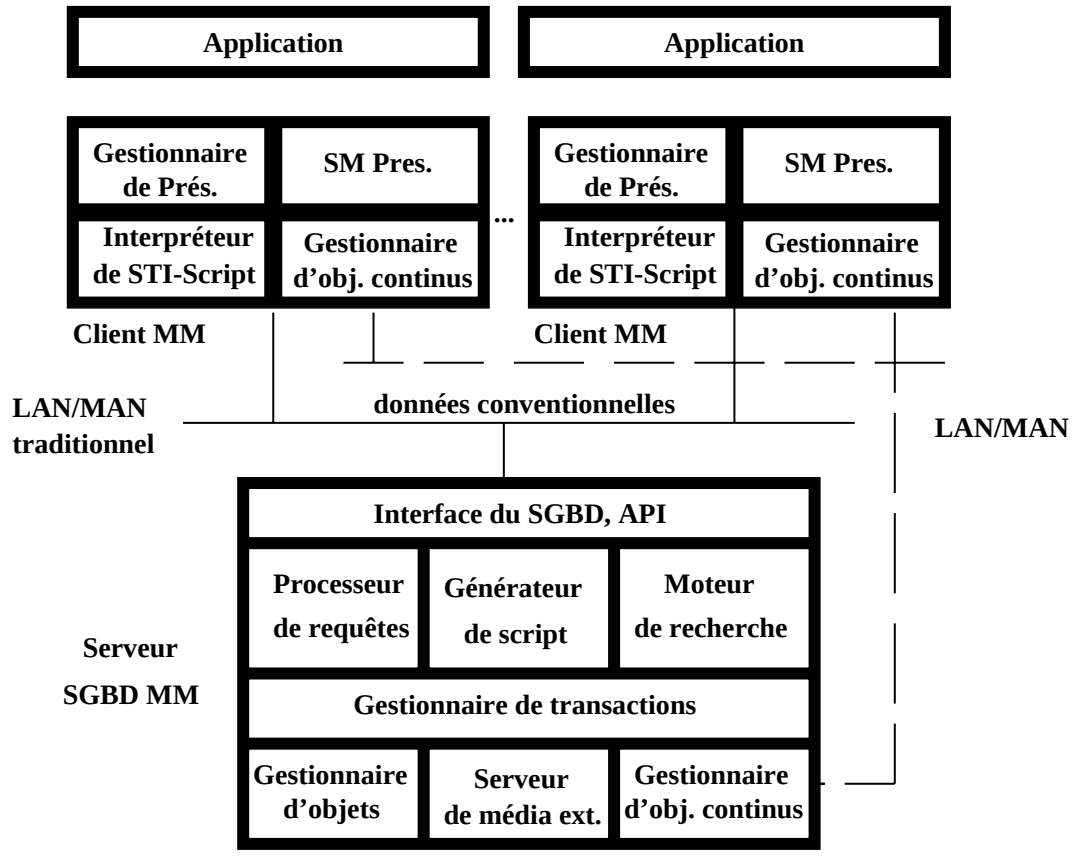

FIG. 2.16 – *L'architecture de référence*

multimédia composée.

- le *gestionnaire d'objets continus* pour la recherche et le traitement des flux de données dépendantes du temps délivrés au serveur.

Au niveau du serveur, les services proposés sont :

- des *services conventionnels* : gestion d'objets pour données conventionnelles, gestion de transactions et traitement de requêtes.

- le *gestionnaire d'objets continus* pour données dépendantes du temps qui sont délivrées en accord avec les paramètres de qualité de service à travers un réseau à grande vitesse vers le site client.

- un support pour des *périphériques de stockage de données multimédias externes* tel que le CD-ROM.

- un support de *recherche sur le contenu.*

Comme nous venons de le voir, les fonctionnalités que doit posséder un SGBD multimédia sont multiples et diverses. De nombreux travaux s'intéressent à ce domaine et montrent l'importance de l'intégration du multimédia dans les systèmes de gestion de bases de données. Mais il reste une caractéristique que nous n'avons pas abordée et qui fait l'objet de travaux récents, il s'agit de l'*interaction avec l'utilisateur*. Les SGBD deviennent multimédias mais aussi interactifs.

2.3.6 Modélisation de l'interactivité

2.3.6.1 Définition de l'interactivité

Les présentations multimédias sont liées à la notion d'opérateur humain. L'utilisateur doit pouvoir interargir au cours du déroulement de la présentation. Ainsi, on obtient différents scénarios possibles pour l'exécution d'une présentation. Si un système de présentations multimédias permet seulement de naviguer suivant les objets multimédias, il s'agit d'une *synchronisation temporelle statique* au moment de la présentation [CC97]. En effet, un système de "navigation seulement" ne peut pas changer les relations temporelles au cours de l'exécution. Par contre, si le système offre plus de liberté pour l'interaction avec l'utilisateur, alors il s'agit d'une *synchronisation temporelle dynamique*. Une interaction avec l'utilisateur est en fait composée d'un événement qui déclenche une action, celle-ci décrivant le scénario lié à cette interaction.

De nombreux systèmes dits interactifs proposent cette notion de scénarios actifs multimédias [PR93, HFK95, VS96, Vaz96, AL96]. Notre étude se base sur la comparaison de trois approches différentes de modélisation de scénarios actifs mais nous verrons que leurs concepts de base sont identiques. Par exemple, [HFK95] fait l'analogie entre la création d'un livre et celle d'une présentation multimédia. Créer un livre consiste à écrire une histoire linéaire qui décrit des événements se produisant dans le temps. Créer une présentation multimédia est beaucoup plus complexe. Par rapport à un livre qui est linéaire, les présentations multimédias contiennent souvent plusieurs médias qui peuvent se produire simultanément ou de façon relative, et l'auteur doit spécifier toutes ces relations. Pour cela, la présentation doit être interactive et différente à chaque exécution. Cette approche s'intéresse donc au comportement temporel des scénarios. Elle étudie comment représenter et modéliser les scénarios multimédias. Le but de ce modèle est de modéliser graphiquement les documents actifs multimédias à l'aide d'une ligne de temps.

Dans l'approche de [VS96, Vaz96], le modèle proposé est une représentation d'ap-

plications multimédias basée sur le concept de scénario. Le scénario est décrit en termes d'événements et d'actions. Le modèle représente tous les événements qui peuvent se produire dans une application multimédia (venant du système, de l'application ou de l'utilisateur). Il propose un schéma pour la représentation d'événements simples ou complexes en regard du spatial, du temporel et du contenu. De plus, il définit un schéma pour représenter les actions dans le cadre des applications multimédias. Ce schéma supporte la composition d'actions dans les domaines du spatial et du temporel. Finalement, le modèle définit un mécanisme intégré pour la description de scénarios d'applications en termes de tuples de scénarios. Un tuple de scénarios définit la réponse de l'application à un événement.

D'autres travaux de [AL96] partitionnent les problèmes de synchronisation dus à l'interaction avec l'utilisateur en deux types : (1) les problèmes de synchronisation dus à des effets spéciaux comme les opérations "skip", "pause", "resume", "reverse" et (2) les difficultés de synchronisation se produisant lors d'une modification par l'utilisateur du déroulement de la présentation. C'est souvent le cas lorsque l'utilisateur espère voir d'autres types de présentations relatives à la présentation en cours, mais ne se trouvant pas dans le déroulement de la présentation.

2.3.6.2 Représentation des événements

Une interaction est bien sûr déclenchée par un événement. Cet événement est souvent décrit comme un événement utilisateur, mais il peut être aussi temporel ou composé.

Une modélisation d'événements est proposée par [VH95, VS96]. La définition de la classe suivante représente les événements génériques :

class **event**

string	*name*
object_id	*subject*
object_id_list	*object*
action_list	*actions*

end

L'attribut `name` est une chaîne de caractères indiquant la sémantique de l'événement. L'attribut `subject` dénote l'objet qui provoque l'événement, tandis que l'attribut `object` dénote la liste des objets qui sont relatifs à l'événement et qui sont affectés

par l'occurrence de l'événement. L'attribut `action` indique les actions déclenchées par l'événement.

Montrons un exemple d'événement à partir de cette définition , l'événement *"l'objet* `A1` *est actif à l'écran"*. Comme décrit dans [VH93], un objet est actif si la méthode `get_status` retourne `TRUE`. L'instance correspondante de l'événement est la suivante :

$$\begin{aligned} name &= A1isActive \\ subject &= A1 \\ object &= null \\ actions &= (get\_status = TRUE) \end{aligned}$$

Un intérêt particulier est porté à la composition d'événements pour représenter des événements complexes qui sont utiles dans les applications du monde réel. Les opérateurs de composition proposés sont : la *conjonction*, la *disjonction* et la *séquence*. Mais aussi les fonctions suivantes : `TIMES(n, e)` qui signifie que l'événement `e` se produit `n` fois consécutives et `IN(e, int)` qui signifie que l'événement `e` se produit durant l'intervalle de temps `int`.

Enfin, l'approche de [VS96] propose une **classification des différents événements** pris en compte :

1. *Evénements utilisateur*
   - Evénements venant de la souris (manipulations de la souris),
   - Evénements venant du clavier,
   - Evénements liés à la voix, ou relatifs au toucher de l'écran,
   - Evénements de contrôle de la présentation : `start`, `pause`, `resume`, `stop`, `fast forward`, `rewind`.
2. *Evénements intra-objet*
   - Invocation d'une méthode,
   - Les changements d'états temporels (actif, suspendu, inactif) ou spatiaux (vu, caché, en couche),
   - Evénements temporels.

3. *Evénements inter-objets*

   - Evénements se produisant si il existe des relations temporelles ou spatiales entre plusieurs objets :
     - dans le cas spatial, par exemple un objet bouge attaché à un autre objet par la contrainte "`meets`",
     - dans le cas temporel, par exemple une déviation entre deux objets continus synchronisés entre eux qui dépasse une certaine limite,
     - ou encore événements spatio-temporels, par exemple "l'objet A s'approche de l'objet B avant 1 heure de l'après-midi".
   - Evénements liés aux applications.
   - Evénements définis par l'utilisateur
     - Evénements liés au contenu du média,
     - Evénements se produisant à un temps spécifique de la présentation d'un objet (la dixième seconde d'une vidéo),
     - Evénements composites.

Leur classification est assez complète, il s'agit bien de tous les événements qui peuvent se produire au cours d'un scénario, mais il est difficile de voir quel type d'actions est déclenché par chaque type d'événements.

2.3.6.3 Modélisation des scénarios

Différentes approches ont été développées pour modéliser un scénario actif multimédia, chacun ayant sa propre philosophie.

- **Modélisation à l'aide d'une ligne de temps**

  Le but du modèle proposé par [HFK95] est de modéliser graphiquement les documents actifs multimédias avec une ligne de temps. Traditionnellement, les médias tels que le texte, les graphiques, l'audio et la vidéo sont présentés suivant leur déroulement dans le temps selon une ligne de temps, comme nous le montre la figure 2.17 avec l'audio "`audio_vidéo_présidents`" et la vidéo "`video_présidents`". Le modèle propose un nouveau type d'objet média appelé "`choix`" qui augmente le pouvoir d'expression du modèle de la ligne de temps. Ce nouvel objet est associé à une structure de données qui contient différents champs:

  - Action_utilisateur: par exemple, un clic souris ou une touche quelconque.

- Région : partie déclenchante de l'action.
- Pointeur_destination_scénario : pointeur sur le scénario approprié.

En résumé, si l'utilisateur réalise l'"action_utilisateur" sur la "région" associée, le "pointeur_destination_scénario" est déclenché.

Le nouvel objet média est placé directement sur la ligne de temps traditionnelle. Par exemple, prenons le scénario de présentation suivant qui présente une vidéo sur les présidents américains Clinton, Bush et Reagan avec un commentaire audio comme le montre la figure 2.17. Les textes affichent le nom et les statistiques de chaque président introduit dans les vidéos. On peut rajouter ensuite des choix permettant à un utilisateur d'en savoir plus sur un des présidents présentés. La structure de données du choix réfère à des sous-scénarios. Il faut noter que les objets `choix` ont une durée.

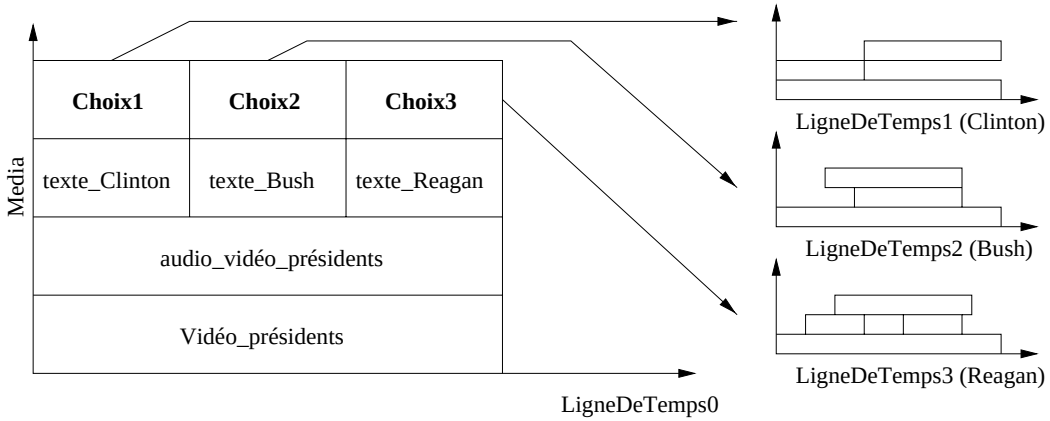

FIG. 2.17 – *Représentation d'un exemple de présentation avec des objets "choix"*

Enfin, ils développent aussi une façon de représenter le scénario actif multimédia graphiquement pour que les auteurs puissent visualiser leur travail en utilisant un arbre de lignes de temps, comme on le voit sur la figure 2.18.

Leur approche est une façon claire et simple pour un utilisateur de spécifier l'interaction, et leur représentation graphique sous forme d'un arbre de ligne de temps, un bon moyen pour visualiser les possibilités offertes à l'utilisateur.

- **Représentation sous forme d'un ensemble de tuples**

Le scénario d'une application multimédia interactive peut être représenté par un ensemble de tuples indiquant la fonction d'actions particulières [VS96]. Les

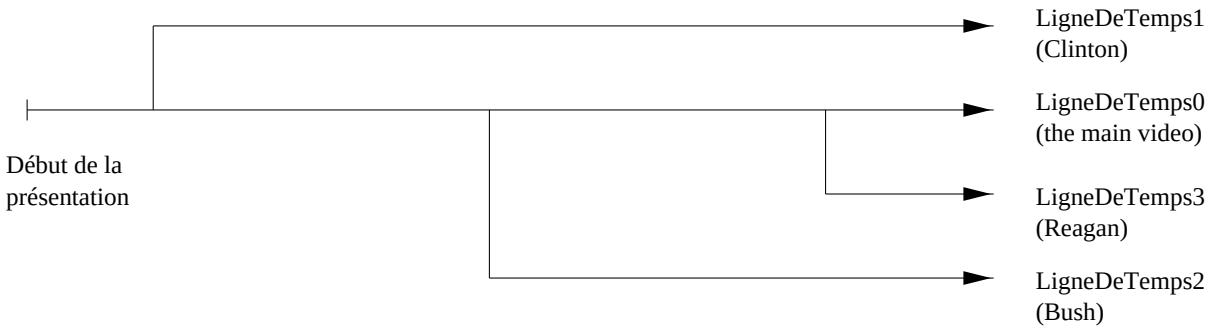

FIG. 2.18 – *Représentation d'un scénario à l'aide d'un arbre de lignes de temps.*

tuples de scénario peuvent être représentés par une classe ayant les attributs suivants :

class **scenario** *tuple*

event_expression	*start_time*
event_expression	*stop_time*
action_expression	*action_list*
content_event_expression	*content_condition*
events	*synch_events*
constraint_expression	*constraints*
boolean	*is_active*

end

- `start_time` représente l'expression de l'événement qui déclenche l'exécution des actions décrites dans `action_list`. L'attribut `stop_time` référence l'expression de l'événement qui termine l'exécution de ce tuple.
- `action_expression` représente la liste des actions et leur composition dans les domaines temporels et spatiaux.
- `content_condition` est une expression indiquant une condition qui fait référence aux contenus des objets multimédias (par exemple, une minute après l'apparition du chien dans la vidéo commencer l'exécution du tuple).
- `synch_events` réfère aux événements produits au commencement et à la fin de l'exécution. Ces événements peuvent être exploités pour la synchronisation.
- `constraints` est un filtre constitué d'un ensemble de contraintes qui peuvent être réalisées au cours de l'exécution du scénario.
- `is_active` indique si le tuple est actif ou non.

Cette approche est basée entièrement sur la notion d'objets dont seuls les attributs décrivent le déroulement du scénario. Il est dommage de modéliser de la dynamique sous une forme statique, alors qu'il est possible de définir des méthodes associées aux objets, celles-ci réalisant la dynamique des objets.

- **Utilisation du concept de lien hypermédia**

Dans certains cas, l'utilisateur espère pouvoir visualiser d'autres présentations relatives à celle qui est en cours, mais qui ne font pas partie de son déroulement. Premièrement, l'approche de [HFK95] est basée sur le concept de *lien hypermédia* utilisant les possibilités de modélisation hiérarchique d'une représentation à l'aide d'un réseau de Pétri. La modélisation sous forme de réseaux de pétri est très utilisée dans le domaine des présentations multimédias. L'interaction de l'utilisateur sur le déroulement de la présentation est ici équivalente à démarrer un nouveau graphe d'une présentation. Se brancher à un autre graphe de présentation est analogue au fait de suivre un lien hypermédia.

Un utilisateur peut décider de voir d'autres présentations à chaque instant durant la présentation multimédia, mais l'autre présentation doit être relative à un de ses objets présentés. Ainsi, à chaque point du réseau de pétri où il y a une branche vers une autre présentation, un noeud est intégré au réseau en parallèle avec un objet (la présentation déclenchée est relative à celui-ci), comme le montre la figure 2.19. Pour représenter ce noeud dans le réseau de pétri, un bouton hypermédia est utilisé. Lorsque l'utilisateur interagit pour naviguer au travers du lien qui lui est proposé, l'exécution du graphe principal est temporairement stoppée et un enregistrement de son état est fait. Alors le sous-réseau représentant la nouvelle présentation est instantié comme le graphe courant.

Deuxièmement, leur approche propose des scénarios basés sur une synchronisation pour effets spéciaux. Le terme de synchronisation pour effets spéciaux sous-entend des besoins de synchronisation lorsque l'utilisateur exécute des opérations spécifiques comme `pause`, `resume`, `reverse`, `forward`, etc.

Opération `Pause/Resume`

Lorsqu'une demande de pause est faite, l'état de la présentation est enregistré (c'est à dire le dernier objet demandé, les objets courants présentés, etc.) et les demandes de recherche d'objets en cours sont stoppées. Puis, au moment de la demande `resume`, le temps au moment de la pause est récupéré et la présentation est redémarrée où elle a été stoppée.

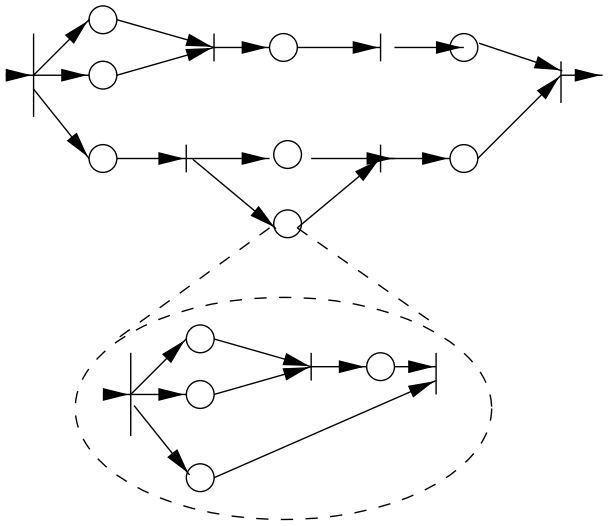

FIG. 2.19 – *Remplacement d'un noeud par un sous-réseau*

Opérations `(Fast-)Reverse` ou `(Fast-)Forward`

Lorsque se produit une demande `reverse`, l'état de la présentation est enregistré et la présentation courante s'arrête temporairement. Pour l'opération `forward`, la direction est opposée. Deux algorithmes sont proposés pour réaliser ces opérations.

Bilan - Les approches proposées s'intéressent plus particulièrement à la modélisation de l'interaction avec l'utilisateur. Seule l'approche de [VH95, VS96] essaie de généraliser en proposant une modélisation objet des événements. Nous pensons qu'il ne faut pas se restreindre à l'étude des événements utilisateurs. Nous allons étudier d'autres types d'événements qui se produisent au cours d'une présentation et voir s'ils peuvent jouer un rôle dans son déroulement en déclenchant des actions particulières.

Ainsi, l'étude de l'interactivité fait apparaître la notion d'exécution d'actions en réaction à un événement. Nous nous sommes ainsi tout naturellement intéressés à la technologie des SGBD actifs pour étudier si elle pouvait être intégrée à celle d'un SGBD multimédia dans le but d'obtenir un *SGBD multimédia actif*. Ainsi, nous pourrions construire des présentations multimédias actives dont l'exécution se ferait à l'aide de règles actives proposées par les SGBD actifs.

2.3.6.4 Utilisation de la technologie des SGBD actifs

Un système de gestion de base de données est dit actif lorsqu'il est capable de détecter des situations significatives et de réagir, sans intervention de l'utilisateur, en exécutant une certaine action. Le comportement d'un SGBD actif s'exprime à l'aide de déclencheurs ou de règles actives (règle E-C-A) [CL92, CHCA94, CHR96]. L'apport d'un SGBD actif à un SGBD multimédia est de pouvoir réagir à n'importe quel type d'événements dépendant de l'utilisateur ou non et d'exécuter toutes sortes d'actions.

Nous avons constaté lors de cette étude que la sémantique d'une règle E-C-A (Evénement-Condition-Action) est très proche de celle d'une interaction. Une interaction comprend un événement qui déclenche une action et celle-ci réalise le scénario décrit par l'interaction.

- **Définition d'une règle active**

  Une règle active est composée de trois parties : la partie *Evénement*, la partie *Condition* et la partie *Action*. La syntaxe générale est : lorsque l'*Evénement* se produit et si la *Condition* est satisfaite alors exécuter l'*Action*.

  La partie *Evénement* d'une règle est un *type d'événement* qui caractérise une classe de faits significatifs pour le déclenchement de la règle. Un type peut par exemple caractériser la modification de l'adresse d'un étudiant ou bien la fin d'une transaction. Dans un SGBD actif, seuls les événements dont les types sont connus seront détectés et gérés. Dans tous les cas, après détection d'un événement, le système détermine les règles déclenchées par cet événement.

  La partie *Condition* d'une règle permet de préciser les situations dans lesquelles il faut exécuter la partie action de la règle.

  La partie *Action* d'une règle est généralement un ensemble d'actions qui sont exécutées de manière séquentielle. Chaque action est spécifiée au moyen du langage de requêtes ou du langage de programmation du SGBD.

- **Le modèle d'événements**

  Nous nous intéressons surtout à l'étude des différents événements qui sont utilisés dans le cadre de notre travail, le but étant de définir un modèle d'événements pour la technologie des bases de données multimédias. Pour cela, nous nous basons sur les études effectuées pour les bases de données actives [CM93, Cou96].

Evénement : Un événement est une occurrence d'un type d'événement auquel on peut associer un point de temps appelé instant d'occurrence.

Modèle d'événements : Pour un système donné, le modèle d'événements spécifie l'ensemble des types d'événements représentables dans ce système. Les événements peuvent être classés en *primitifs* et *composites*.

Evénements primitifs : les événements primitifs sont eux-mêmes classés en quatre catégories : les *événements internes*, les *événements temporels*, les *événements utilisateurs*, les *événements systèmes*.

1. Les *événements internes* sont constitués par tous les événements émanant directement du SGBD. Ils sont liés pour la plupart aux manipulations des données et des transactions. Les événements internes sont basés sur les *transitions* entre les différents états d'une base de données.
2. Les *événements temporels* dénotent un point déterminé dans le temps. Par exemple l'événement "`10 minutes après le début de la présentation`", cet événement se produit à un temps d'une durée de 10 minutes par rapport au temps zéro (qui est le temps de démarrage de la présentation dans notre étude).
3. Les *événements utilisateurs* ne sont pas nécessairement liés à des opérations sur une base. Ils ne peuvent pas être directement détectés par le système et doivent être signalés explicitement dans le code des applications. Ces événements sont les plus utilisés pour l'interaction.
4. Les *événements externes* ne sont liés à aucun phénomène opératoire du SGBD mais proviennent de l'environnement de celui-ci et peuvent être capturés par le système de règles. Des exemples d'événements sont l'arrivée d'un courrier électronique ou l'arrivée de relevés en provenance d'un thermomètre, une alarme, etc.

Le tableau 2.2 identifie les types d'événements primitifs du système actif NAOS qui permet la définition et l'exécution de règles actives pour les applications O_2 [CCS94, CC96b, Col97]. NAOS a, en effet, été développé en considérant le SGBD O_2 comme noyau de gestion des données et d'exécution des applications. Il s'intègre dans l'architecture modulaire de ce système et constitue donc un

nouveau composant utilisable pour le développement des applications, à côté des autres composants élémentaires que sont O_2C, l'interface C++ et O_2SQL.

type d'opération	**description**
CREATE	Création d'une entité
DESTROY	Destruction d'une entité
RETRIEVE	Accès à une entité
INSERT	Insertion dans une collection
DELETE	Suppression dans une collection
UPDATE	Mise à jour d'une entité
ATTACH	Attachement à une racine persistante
DETACH	Détachement d'une racine persistante
METHOD_BEGIN	Appel d'une méthode d'un objet
METHOD_END	Fin d'exécution d'une méthode
TRANSACTION_BEGIN	Début d'une transaction
TRANSACTION_VALIDATE	Validation d'une transaction
TRANSACTION_COMMIT	Commit d'une transaction
TRANSACTION_ABORT	Abandon d'une transaction
PROGRAM_BEGIN	Début d'un programme
PROGRAM_END	Fin d'un programme
APPLICATION_BEGIN	Début d'une application
APPLICATION_END	Fin d'une application

TAB. 2.2 – *Les événements dans le système NAOS*

Evénements composites : un type d'événement composite est défini par une expression incluant des types d'événements primitifs ou composites et des opérateurs de composition dont les plus répandus sont la *disjonction*, la *conjonction* et la *séquence* [CKAK94, CC96c]:

Les événements composites définis dans NAOS utilisent les opérateurs suivants :

Soit {E1, E2, . . . , En} des types d'événements et {e1, e2, . . . , en} leurs instances respectives. Les opérateurs disponibles dans NAOS sont alors définis par :

Disjonction - Un type d'événement caractérisant la disjonction de deux événements e1 et e2 est de la forme : E1 — E2. Une instance de ce type se produit si e1 ou e2 (ou les deux) se produisent.

Conjonction - Un type d'événement caractérisant la conjonction de deux événements e1 et e2 est de la forme : E1 & E2. Une instance de ce type se produit si e1 et e2 se produisent, dans n'importe quel ordre.

Séquence - Un type d'événement caractérisant la séquence de deux événements e1 et e2 est de la forme : E1 , E2. Une instance de ce type se produit si le **dernier** événement qui compose e1 se produit avant le **dernier** événement qui compose e2.

Séquence stricte - Un type d'événement caractérisant la séquence stricte de deux événements e1 et e2 est de la forme : E1 ; E2. Une instance de ce type se produit si le **dernier** événement qui compose e1 se produit avant le **premier** événement qui compose e2.

Négation - Un type d'événement caractérisant la négation d'un événement e1 est de la forme : !E1. Une instance de ce type se produit si e1 ne se produit pas dans un intervalle de validité.

Disjonction stricte - Un type d'événement caractérisant la disjonction stricte (exclusive) de deux événements e1 et e2 est de la forme : E1 $\bigwedge$ E2. Une instance de ce type se produit si e1 ou e2 (mais pas les deux) se produit dans un intervalle de validité.

Intervalles de validité - Un intervalle de validité est défini par ses deux bornes qui sont des types d'événements. Pour les opérateurs dits *négatifs* (Négation et Disjonction stricte), l'intervalle de validité décrit la période pendant laquelle il faut détecter l'*absence* d'un événement. Par défaut cet intervalle est [*début_transaction*, *fin_transaction*], mais il peut être différent selon les opérateurs utilisés dans l'expression.

Exemple : Dans le type d'événement composite `e1,!e2,e3` l'intervalle de validité de `!e2` est `[e1,e3]`. Cela signifie que la non-détection de `e2` se fera entre `e1` et `e3` ce qui revient à dire que lors de la détection du type d'événement composite, on s'assurera qu'aucune occurrence de `e2` n'est intervenue entre l'occurrence de `e1` et celle de `e3`.

- **Traitement des événements**

  Lors de sa définition, une règle R est associée à un type d'événement E. Lorsque plusieurs occurrences (ou instances) de E sont produites dans une même unité de production, il existe deux stratégies de prise en compte de ces événements qui influent sur le déclenchement de R :

  1. la première stratégie consiste à traiter les événements instance par instance et donc à exécuter R autant de fois qu'il y a d'événements déclenchants, on parle alors de *sémantique d'instance*.
  2. la seconde stratégie consiste à n'exécuter qu'une fois pour l'ensemble des événements, on parle alors de *sémantique ensembliste*.

  Dans les systèmes à objets, la sémantique d'instance est la plus utilisée : Exact, Ode, Reach, Samos, Beeri & Milo. Dans NAOS [CCS94], on trouve les deux sémantiques.

  Instant de déclenchement : Le mode de traitement des événements détermine l'*instant de déclenchement d'une règle*, c'est-à-dire l'instant associé à la transition de l'état *déclenchable* vers l'état *déclenché*. Pour une règle ayant une sémantique d'instance, l'instant de déclenchement est l'instant d'occurrence de l'événement déclenchant. Pour une règle ayant une sémantique ensembliste, l'instant de déclenchement de la règle est l'instant d'occurrence de son premier (dans le temps) événement déclenchant.

  Occurrence d'une règle : Une *occurrence d'une règle* est représentée par un triplet $< r,\ t,\ e >$ dans lequel r est la règle considérée , t est son instant de déclenchement et e son environnement (d'évaluation et d'exécution). La notion d'occurrence d'une règle est importante en ce sens qu'en fonction du mode de traitement des événements et du mode de consommation des événements (voir paragraphe suivant), il est possible qu'à un point de considération, il existe plusieurs occurrences d'une même règle (la règle a été déclenchée plusieurs fois).

- **Consommation des événements**

  Comme nous venons de le voir, une règle est susceptible d'être exécutée plusieurs fois dans une même unité de production. Chaque événement déclenchant e_i est conservé jusqu'à la première exécution de la règle qu'il a déclenché. Un événement est donc pris en compte au moins une fois au cours de cette exécution. En ce

qui concerne les éventuelles exécutions ultérieures de cette même règle, deux politiques de traitement de l'événement peuvent être adoptées :

1. *Consommation :* l'événement e_i est consommé par la première exécution de la règle et ignoré dans les exécutions ultérieures.
2. *Préservation :* après avoir été pris en compte une première fois, l'événement e_i est préservé et donc pris en compte dans les exécutions ultérieures de la règle.

La consommation des événements peut être *locale*. Dans ce cas, l'événement e_i peut déclencher d'autres règles non encore considérées. La consommation peut également être *globale*. Dans ce cas, e_i est ignoré dans les exécutions ultérieures de la règle mais aussi dans celles de toutes les autres règles (e_i aura donc provoqué l'exécution d'une seule règle).

Remarque - Nous montrerons, au cours de cette thèse, que la technologie des *SGBD actifs* peut jouer un rôle dans le développement des futurs *SGBD multimédias* et plus particulièrement dans l'exécution de l'interaction avec l'utilisateur et de scénarios actifs.

2.4 Conclusion

Dans ce chapitre, nous avons tout d'abord étudié les différentes fonctionnalités que doit offrir un SGBD multimédia comme (1) l'intégration des *aspects temporels et spatiaux* liés aux données multimédias; (2) la *représentation* et la *modélisation* des données multimédias; (3) la *recherche de l'information multimédia*, c'est à dire pouvoir interroger les données multimédias sur leur contenu en utilisant leur modélisation qui fournit des mécanismes d'indexation.

Après cette étude, nous pouvons affirmer que la technologie des bases de données apporte un plus pour la manipulation des données multimédias, et plus particulièrement celle des bases de données à objets. En effet, elles permettent d'utiliser tous les atouts de l'approche objet comme la manipulation d'objets complexes, l'utilisation de méthodes pour associer un comportement aux objets multimédias ou enfin la réutilisabilité des objets. Un des avantages des bases de données est aussi la facilité pour faire le lien entre la structure des bases de données et les concepts multimédias tels que les aspects temporels ou spatiaux. Les bases de données offrent en plus des langages

de requêtes déclaratifs puissants pour l'interrogation des données multimédias. Par contre, il faut redéfinir l'architecture des SGBD pour pouvoir intégrer, par exemple, les services d'un gestionnaire de présentations ou d'un serveur continu, etc.

Nous nous sommes aussi intéressés aux travaux sur la modélisation de l'interactivité qui apporte des solutions à l'intégration de l'interaction avec l'utilisateur dans le processus de présentation des données multimédias. Ils proposent entre autres une modélisation des scénarios qui permettent diverses présentations des données multimédias et intègrent l'interactivité avec l'utilisateur. Notre étude des systèmes interactifs nous a conduit vers celle des systèmes actifs qui peuvent réagir à des événements et, selon une condition, exécuter une action. Cette technologie des bases de données actives jouera un rôle dans le cadre du développement des bases de données multimédias.

Cette étude des systèmes multimédias existants montre la diversité des approches mais aussi les nombreux besoins pour le développement des SGBD multimédias. Les thèmes de recherche sont variés et en plein essor. Les chapitres suivants montrent nos travaux sur la modélisation de présentations multimédias et de leur dynamique. A partir des concepts de ces deux modèles, nous décrirons notre réalisation d'un SGBD multimédia.

Chapitre 3

Modèle de présentations multimédias

Ce chapitre présente le modèle STORM [Adi95, Adi96, AM97] qui a servi de base à nos propositions. Ce modèle propose des solutions pour la modélisation et la gestion des données basées sur le temps et permet de décrire des présentations séquentielles ou parallèles, de les construire, les mettre à jour et les interroger. Ces présentations sont stockées dans la base de données au même titre que les objets qui servent à les construire.

Une présentation complexe est composée de différents objets à présenter qui possèdent des contraintes temporelles (délai, durée) et des contraintes de synchronisation (l'un après l'autre, en parallèle, etc.). Cette approche considère chaque présentation comme un objet sur lequel des opérations spécifiques peuvent être définies (interrogation, mise à jour, et exécution). L'exécution d'une présentation doit pouvoir se faire en respectant les contraintes de synchronisation.

La Section 3.1 présente les différents objets multimédias manipulés et stockés dans la base et introduit la notion de présentation multimédia. La Section 3.2 décrit les aspects temporels des différents objets qui composent une présentation et introduit le concept d'*Ombre Temporelle* constituée d'un délai et d'une durée qui forment les apsects temporels liés aux objets. La Section 3.3 décrit les *objets STORM* qui correspondent aux présentations des objets de la base. La Section 3.4 présente les notions de présentations séquentielles ou parallèles. La Section 3.5 définit la cohérence temporelle qui exprime que les objets dans une présentation doivent obéir à des contraintes temporelles spécifiques. La section 3.6 présente le concept de présentations intention-

nelles, il s'agit de présentations basées sur des requêtes. Enfin, la section 3.7 propose un exemple de présentation multimédia et montre sa création par l'utilisateur.

3.1 Objets multimédias et Présentations

En général, les SGBD à objets manipulent les types atomiques suivants (statiques) : `integer`, `real`, `boolean`, `bits`, `char`, `string`, `Date`, `Time`. `Atomic` dénote l'union de tous ces types. Bien qu'il soit possible de présenter un entier ou une chaîne de caractères durant un temps donné, une attention spéciale est portée aux types prédéfinis suivants `Image, Text, Audio, Video`. Ces types fournissent une indépendance physique et aucune hypothèse particulière n'est faite sur les serveurs des différents médias. Les objets peuvent être intégrés dans un seul système ou être gérés par différents serveurs interconnectés. Le concept d'encapsulation assure ici l'indépendance des données par rapport au stockage physique.

Les objets sont créés et stockés dans la base indépendamment de leur présentation. Par exemple, il peut s'agir d'une base d'images où différents attributs externes comme le titre, la date, le sujet sont associés au contenu de l'image.

Les modèles de données à objets traditionnels ne prennent pas en compte les aspects temporels pour combiner et synchroniser différents objets à présenter à l'utilisateur. Le modèle propose d'ajouter une dimension temporelle à toute description statique et structurelle. Par exemple, allouer à chaque image un temps donné, ou si des commentaires vocaux sont associés aux images, présenter chaque couple (son, image) en parallèle immédiatement suivi du second couple, etc. Une **présentation** est définie comme un objet STORM (cf. 3.3) où un temps spécifique est alloué à chaque objet à présenter. La synchronisation (en séquence et en parallèle) entre les objets est également exprimée. Un exemple classique d'une telle présentation est un diaporama. Ainsi si `i1`, `i2`, `i3` sont des instances d'images et `a1`, `a2`, `a3` des instances de sons, la valeur suivante est une liste de couples (image, son): `((i1, a1), (i2, a2), (i3, a3))` constituant le diaporama.

3.2 Aspects temporels des présentations

Le modèle temporel choisi est basé sur les notions déjà largement acceptées et utilisant les intervalles et les opérateurs associés [All83, AN86, LG91b, LG93]. Il faut

également exprimer les durées et les délais pour les objets à présenter. Pour chaque présentation, plusieurs intervalles de temps peuvent être définis. Un intervalle est défini de la façon suivante : soit un ensemble `S` partiellement ordonné ($\leq$) et deux éléments `a` et `b` de `S`, l'ensemble $\{x|\ a \leq x \leq b\}$ est un intervalle de `S` (noté [a,b]) [Gha94].

Si on fait correspondre à `S` un ensemble d'instants, on peut parler d'intervalles temporels. Dans le modèle STORM, *chaque intervalle correspond à la présentation d'un objet* (une image ou une musique). Le début et la fin d'un intervalle sont des **temps logiques** qui correspondent en réalité à un temps physique durant la présentation effective à l'utilisateur. Pour un intervalle [a,b], a et b correspondent à des événements spécifiques qui sont soit déclenchés manuellement (utilisateur) ou automatiquement (système). La granularité du temps est de une seconde et la longueur des intervalles correspond à une durée exprimée en secondes.

Différentes relations ont été définies sur les intervalles de temps : `before, meet, equal, overlap, during, start, finish` [All83]. Il s'agit de relations binaires, mais elles peuvent être facilement étendues à des relations n-aires [LG93]. Les relations dites séquentielles combinent les intervalles qui partagent la même ligne de temps (exclusion mutuelle), se produisant l'un après l'autre avec (`before`) ou sans délai (`meet`) entre eux. Les relations dites parallèles relient des intervalles qui ont leur propre ligne de temps. Dans le modèle, ces relations sont utilisées pour composer et synchroniser les objets multimédias dans les présentations.

Pour la modélisation des données multimédias, et pour une raison de simplicité, le texte (chaîne de caractères de longueur arbitraire) et l'image (de type Bitmap) sont considérés comme des données multimédias **statiques**. Le son et la vidéo sont considérés comme **dynamiques** ou données éphémères. Statique signife qu'il n'existe pas un temps spécifique associé à une image. Elle peut être affichée dix minutes ou dix secondes. Les données dynamiques sont dépendantes du temps de façon explicite. Une vidéo est affichable pendant un temps spécifique à une cadence de 25 ou 30 images par seconde. La durée d'une séquence audio ou vidéo peut être soit stockée avec l'objet, soit calculée (par exemple sur la base du nombre d'images), mais elle est considérée comme une propriété inhérente de l'objet.

3.2.1 L'Ombre Temporelle

Chaque objet multimédia doit pouvoir être présenté à l'utilisateur, mais pour un même objet, on peut en vouloir des présentations différentes. Par exemple, si x est une

image, elle peut être présentée 3 minutes dans une présentation et 30 secondes dans une autre. De plus, avant de percevoir réellement un objet multimédia, il est intéressant de définir un délai. Par exemple, le système affiche une icône montrant qu'une image est prête à être affichée. Là encore pour un objet donné le délai peut varier d'une présentation à une autre. Ceci conduit à la notion d'**ombre temporelle** associée à chaque présentation d'objet. Ainsi, selon l'éclairage temporel que l'on applique à un objet à présenter, l'ombre temporelle est différente.

Une ombre temporelle est composée de deux intervalles : un délai et une durée représentés sur la figure 3.1. Pour chaque objet x, le début du premier intervalle correspond au début de la présentation de x (noté ici `present(x)`). Le début du second intervalle est `present(x)+delay(x)`, noté `begin(x)`. La fin de la présentation se produit à `present(x)+delay(x)+duration(x)`, noté `end(x)`.

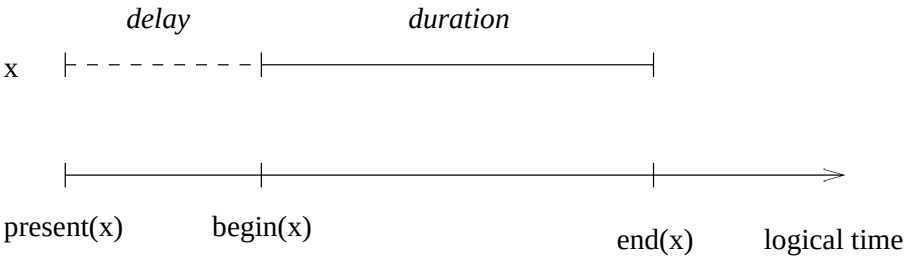

FIG. 3.1 – *Les deux intervalles de temps pour présenter un objet x*

La durée (notée duration(x)) est le temps (en secondes) durant lequel l'objet est perçu par l'utilisateur. Par exemple, la durée d'une image (ou d'un entier) est le temps durant lequel l'image est affichée à l'écran, tandis que pour un objet audio, il s'agit du temps durant lequel le son est joué. Une **durée** a soit une valeur illimitée ou indéfinie (qualifiée de `free` dans le modèle), soit limitée (qualifiée de `bound`). Pour les objets statiques, la durée est illimitée par défaut. Ceci signifie qu'une fois une image affichée, l'utilisateur a la responsabilité de l'effacer. Naturellement, il est possible d'allouer un temps fixe (5 minutes par exemple) durant lequel l'image est affichée et ensuite automatiquement effacée. Par contre pour les objets dynamiques ou éphémères comme l'audio ou la vidéo, la durée est fixée par la nature même de l'information à présenter, si on ne veut pas de distorsion.

Dans une présentation, un **délai** est associé à chaque donnée. Pour chaque objet x, delay(x) est le temps (en secondes) avant d'observer x. En d'autres termes, il y a un temps d'attente avant de présenter (jouer) l'objet au niveau de l'interface, par

exemple, attendre dix secondes avant de jouer un commentaire audio. Ici aussi, il existe un délai illimité (`free`) ou limité (`bound`). Un délai limité a une signification évidente et vient juste d'être décrit. La notion de délai illimité nécessite cependant certaines explications. Par définition, un délai illimité signifie que l'on est en attente "pour toujours" ! Le modèle propose d'utiliser ce concept pour exprimer le fait que le système est prêt à présenter un objet, mais attend qu'un événement se produise, typiquement une action de l'utilisateur. Par exemple, le système affiche un icone montrant qu'un objet vidéo est prêt à être joué (une petite TV) avec des boutons spécifiques tels que "play", "stop", "backward", "forward" et attend jusqu'à ce que l'utilisateur choisisse, par exemple le bouton "play".

3.3 Les objets STORM

Un objet STORM (ou `SO`) est une présentation, définie par un **quadruplet (i, d, δ, c)** où `i` est un identificateur. La durée **d** et le délai δ constituent l'`Ombre Temporelle`. L'information à présenter est le **contenu c** qui peut être soit monomédia, soit composé de différents objets en utilisant des opérateurs séquentiels et/ou parallèles.

Les règles suivantes définissent récursivement les objets STORM :

- **Règle 0** : **(i, d, δ, nil)** est un `SO`[1].
- **Règle 1** : **(i, d, δ, c)** est un `SO` où c est une instance de `Atomic, Image, Text, Audio, Video`.

  Deux principaux opérateurs temporels, séquence (`seq`) et parallèle (`par`) sont utilisés pour construire des présentations. Ils expriment les contraintes temporelles et de synchronisation pour présenter plusieurs objets ensembles.

  Si s_1, s_2,... s_n (n$\geq$1) sont des `SO`, alors

- **Règle 2 : (i, d, δ, seq(s_1, s_2,...s_n))** est un `SO` (une présentation séquentielle).
- **Règle 3 : (i, d, δ, par(s_1, s_2,...s_n))** est un `SO` (une présentation parallèle).

  Du fait de la nature récursive de ces définitions, une présentation (ou un objet `SO`) combine en séquence ou en parallèle plusieurs présentations des objets qui la composent. Chaque s_i mentionné plus haut a sa propre `Ombre Temporelle`

1. Une ombre temporelle est associée à nil qui peut changer en fonction du contexte où nil est utilisé.

notée (δ_i, $\mathtt{d}_i$). Pour chaque présentation séquentielle ou parallèle, le délai δ est le temps avant de commencer toute la présentation, tandis que la durée `d` dépend des durées $\mathbf{d}_i$ et des délais δ_i, i ϵ [1..n]. Par exemple, si on a la présentation `seq(`$\mathtt{s}_1$`,` $\mathtt{s}_2$`)` avec $\delta_1 = \delta_2 = 0$, alors $\mathtt{d} = \mathtt{d}_1 + \mathtt{d}_2$.

- **Règle 4 : Qualification des présentations**

  Pour exprimer la synchronisation entre les objets, les opérateurs temporels génériques `seq` et `par` peuvent être qualifiés. Ils sont notés par `seq_`α (où α est `meet` ou `before`) et `par_`β (où β est `equal, start, finish, during, overlap`). Ces opérateurs sont présentés sur la figure 3.2 et sont décrits par la suite. La plupart du temps, il s'agit de présentations de deux objets mais les présentations peuvent être n-aires.

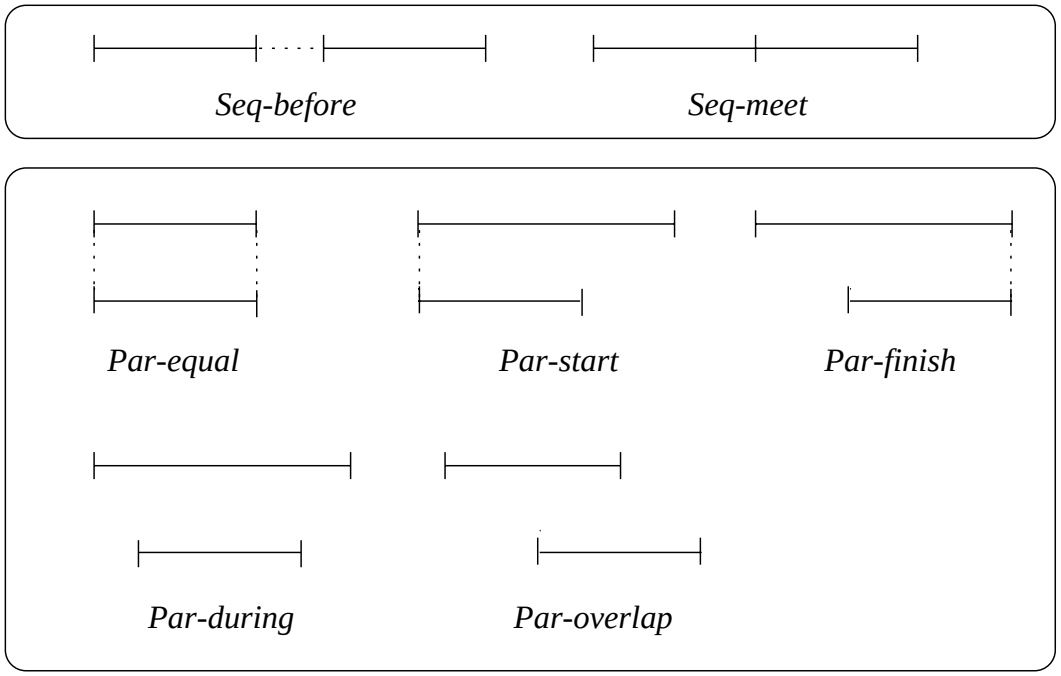

FIG. 3.2 – *Les opérateurs temporels*

Ces notions de présentations séquentielles et parallèles sont logiques, dans le sens où on ne prend pas en compte l'existence de canaux pour les objets correspondants. Par exemple, dans les présentations parallèles, l'existence de canaux physiques différents permet un vrai parallélisme. Cette approche fournit un bon niveau d'indépendance car elle se concentre sur les aspects temporels et de synchronisation entre les objets. La correspondance entre une présentation STORM

et les canaux physiques doit être faite au niveau de l'interface utilisateur.

Une autre remarque importante est que les présentations se réfèrent aux objets multimédias, mais ne les "contiennent" pas. Ceci est important car ces objets ont, en général, une grande taille et ne doivent pas être dupliqués. Avec cette approche, différentes présentations peuvent se partager le même objet.

3.4 Présentations séquentielles et parallèles

L'opérateur `seq` construit une présentation séquentielle. Par exemple, si `m1, m2` sont des présentations de musiques (il s'agit donc bien d'objets `SO`) et `p1, p2` des présentations d'images, on peut avoir: (i) `seq(m1,m2)` pour une séquence musicale; (ii) `seq(m1,p1)` où l'affichage de `m1` précède `p1`; (iii) deux séquences de `p1` et `p2` avec différentes TS, `s1= seq(p1,p2)` avec TS1=((d_{11}:10, δ_{11}:F), (d_{21}:20, δ_{21}:F)), et `s2= seq(p1,p2)` avec TS2=((d_{12}:F, δ_{12}:30), (d_{22}:F, δ_{22}:40))[2].

- `seq_meet(`o_1`,`o_2`)`

La relation **meet** impose qu'il n'y ait pas de délai entre les objets et qu'ils soient joués l'un après l'autre. Si toutes les durées sont `free`, il y a deux cas :
- présenter o_1[3], puis attendre un événement externe (par exemple, venant de l'utilisateur), puis traiter o_2, etc.;
- fixer d_i pour chaque o_i. Chaque durée peut être différente ou non, l'une peut être calculée à partir de l'autre ($d_{i+1} = f(d_i)$).

Si les durées sont `bound`, alors chaque objet a sa propre durée et sera présenté en conséquence.

Supposons des objets de type `Audio a1...a5`. Si on veut les jouer en séquence sans délai, il faut construire l'objet : `seq_meet (a1, a2, a3, a4, a5)`.

Dans une présentation, il est possible de répéter un objet donné. Considérons, par exemple, la séquence suivante `seq_meet (m12*,v10)` où `m12` est un objet audio (une musique) et `v10` une vidéo. Tous les objets ont des durées `bound`, mais `m12` est répété[4], le résultat est une durée `free` pour le premier opérande. La musique doit

2. *Les durées sont données en secondes et F veut dire free.*
3. Un objet o_i est un objet `SO`
4. '*' est un opérateur de répétition, signifiant "pour toujours".

alors être stoppée pour commencer la présentation de la vidéo. Ceci est différent de `seq_meet(m12,v10)`.

- `seq_before(`o_1`,`o_2`)`

La relation **before** introduit un délai non nul entre chaque objet. Ce délai peut être le même pour tous les éléments ou peut être différent. Après un délai δ_1, l'objet o_1 est présenté durant d_1 et puis effacé. Ensuite il faut attendre pendant un délai δ_2 avant de présenter l'objet o_2, etc. La discussion sur la liste des durées est similaire au cas précédent. La différence ici est que la liste des délais peut être `free`, `partiellement bound` ou `bound`. Un délai `bound` est directement manipulé par le système. Un délai `free` signifie que le système attend avant de jouer le prochain objet, aussi il est nécessaire de prendre en compte les événements utilisateurs.

Dans une présentation parallèle, différents objets peuvent être perçus simultanément, chaque objet ayant sa propre ligne de temps. Dans le cas général, il n'y a pas de contraintes de synchronisation prédéfinies spécifiques entre les objets. Pour imposer des contraintes de synchronisation entre les objets, l'opérateur `par` devra être qualifié: `par_equal, par_during, par_start, par_finish, par_overlap`. Certains qualifieurs nécessitent que les délais soient `bound` et nuls (par exemple `par_start`), tandis que pour d'autres, les délais ne doivent pas être nuls (`par_overlap`).

- `par_equal(`o_1`,`o_2`)`

Cette relation impose que toutes les durées d_i soient les mêmes et que tous les délais soient nuls. La liste des durées peut contenir zéro, une ou plusieurs valeurs `bound`. Il y a trois possibilités[5] pour (d_1, d_2) :
- (`Free, Free`) : o_1 et o_2 commencent simultanément et dès que l'un est arrêté, il faut arrêter l'autre;
- (`Free, Bound`) : il faut considérer ici que la durée `bound` de o_2 est affectée à o_1 et la présentation résultante est `bound`. Il faut noter que l'utilisateur a la permission de stopper o_2 ou o_1 à chaque instant. Cependant, stopper l'un des deux déclenchera automatiquement la fin de l'autre;
- (`Bound, Bound`) : ce cas est seulement permis si d_1=d_2 , autrement on a une incohérence temporelle au niveau de l'objet SO (voir Section 3.5). Il faut noter qu'il est possible d'étendre ou de diminuer o_1 ou o_2 pour rendre d_1 et d_2 égales.

Par exemple, considérons l'objet `music1` (type `Audio`) qui sera joué en musique de

5. Dans ce cas, `par_equal(`o_1`, `o_2`) = par_equal(`o_2`, `o_1`)`.

fond lorsque une séquence de (trois) images `p1, p2, p3` sera présentée, chacune d'elle associée avec un commentaire audio `a1, a2, a3`[6]. La valeur suivante comme contenu d'un `SO` exprime la synchronisation désirée et la figure 3.3 nous présente graphiquement le déroulement de cette présentation au cours du temps :

```
par_equal(music1*,
          seq_meet(par_equal(p1,a1),
                   par_equal(p2,a2),
                   par_equal(p3,a3)))
```

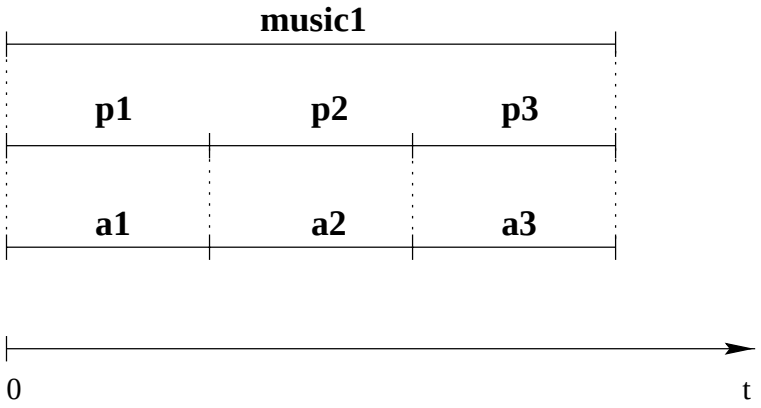

FIG. 3.3 – *Déroulement graphique de la présentation*

Le `par_equal` externe a deux paramètres : une musique est répétée (`Free`) et une séquence (sans délai) de flots parallèles qui sont de type (`Free`, `Bound`). Chaque image est affichée (`Free`) avec son commentaire associé (`Bound`). La contrainte `par_equal` entre les présentations d'images et de sons s'obtient en allouant la durée de chaque objet audio à la durée de l'image correspondante. La séquence a donc une durée `Bound`, ainsi le `par_equal` externe est de type (`Free`, `Bound`) et la musique de fond est stoppée dès que la présentation est terminée (voir Section 3.5).

- `par_during(`o_1`,`o_2`)`

La relation **during** impose que $d_1 > d_2$ ou en d'autres termes que o_{i-1} ne peut pas être stoppé si o_i est encore actif. Le modèle l'exprime par des contraintes associées à l'opérateur `par_during` appliqué aux objets. Dans `par_during(`o_1`, `o_2`)`, o_1 commence

6. Il est admis que la station de travail utilisateur possède deux canaux audio.

en premier et après un délai δ_2, o_2 démarre. Cependant, la durée d_2 associée à o_2 doit être telle que $d_2 + \delta_2 < d_1$, et ainsi pour tous les objets.

Si d_1 et d_2 sont `bound`, il peut être intéressant de centrer de façon temporelle l'objet o_2 se présentant au milieu de o_1. Le même effet peut être obtenu en modifiant le délai de o_2 dans son `Ombre Temporelle`: $\delta_2 = (d_1\text{-}d_2)/2$.

- `par_start(`o_1`,`o_2`)`, `par_finish(`o_1`,`o_2`)`

La relation **start** impose seulement que tous les o_i soient affichés en parallèle sans contraintes sur leurs durées respectives. On peut observer que `par_start` impose seulement que différents objets commencent simultanément tandis que `par_equal` correspond aux objets qui commencent et finissent ensemble.

La relation **finish** doit considérer les durées respectives des données mais aussi les délais possibles entre le début de o_1 et le début de o_2, par exemple. La contrainte doit assurer que si l'objet o_1 est stoppé (à cause d'un "time-out" ou par action de l'utilisateur), o_2 s'arrête en même temps. Considérons `par_finish(`o_1`, `o_2`)` et admettons que d_1 est `free` et d_2 est `bound`. Dans ce cas-là, d_1 est égale à $\delta_2 + d_2$.

- `par_overlap(`o_1`,`o_2`)`

Une telle présentation signifie que o_1 commence en premier, suivi après un délai (non nul) par o_2, etc. La contrainte est que la fin de o_1 se produira avant la fin de o_2. Analysons les cas où certaines durées sont `Free` (F) et d'autres `Bound` (B). Considérons les deux objets o_1 et o_2, et leur durée respective d_1 et d_2. Il existe quatre cas, (1) : (F, F), (2) : (F, B), (3): (B, F), (4): (B, B). La contrainte est facilement vérifiée pour les cas (1) et (4). Le cas (2) est interdit car le résultat est une relation `during`. Le cas (3) est permis mais o_2 ne peut être stoppé avant la fin de o_1.

Concernant `overlap`, un autre cas intéressant est celui où l'on veut construire une transition entre deux présentations d'objets p_1 et p_2. On suppose qu'avant la fin de p_1, p_2 démarre et qu'ils se recouvrent durant le temps de la transition. Dans le cas de durées `bound` sans action de l'utilisateur, il est facile de calculer δ_2 pour démarrer p_2 en conséquence.

Avec la liste de couples (image, son) décrite au paragraphe 3.1 : `((i1, a1), (i2, a2), (i3, a3))`, on construit une présentation parallèle : `seq_meet(par_equal(i1, a1), par_equal(i2, a2), par_equal(i3, a3))`.

Cas général

Dans le cas général, l'`Ombre Temporelle` pour des présentations séquentielles et parallèles peut être calculée comme cela est indiqué en table 3.1. Supposons que $\Theta(o_1, o_2)$ soit une présentation temporelle où Θ peut être l'un de sept opérateurs (deux séquentiels et cinq parallèles). Les durées respectives de o_1 et o_2 sont d_1 et d_2 (`free` et `bound` sont notés respectivement par F et B). Pour tous les opérateurs, deux durées `free` ou deux durées `bound` résultent respectivement en durées `free` ou `bound`. Cependant, quand une durée est `free` et l'autre `bound`, le résultat dépend de l'opérateur. Les opérateurs `par_equal` et `par_finish` ont une sémantique particulière, le résultat est `bound`. Pour tous les autres, le résultat est `free`.

durée d1 d2	F	B
F	F	par_equal ou par_finish: B autre: F
B	par_equal ou par_finish: B autre: F	B

TAB. 3.1 – *Combinaison des durées Free (F) et Bound (B)*

3.5 La cohérence temporelle

La cohérence temporelle exprime que les objets dans une présentation doivent obéir à des contraintes temporelles spécifiques. On distingue les contraintes statiques et dynamiques. Les contraintes temporelles statiques expriment la cohérence au moment de la création, tandis que les contraintes dynamiques doivent être vérifiées au cours de la présentation.

Considérons deux objets audio a1 et a2 avec pour durées respectives d1 et d2 (`bound`) et pour délais $\delta 1$ et $\delta 2$ (`bound`) et supposons que nous voulons construire l'objet STORM `par_overlap(a1,a2)`. A la création, pour que l'objet soit cohérent il faut vérifier que $d_2 + \delta_2 > d_1$. Notons que cette contrainte peut être vérifiée parce que les durées sont `bound`. Lors de la présentation, il faut contrôler la contrainte suivante: end(a2) > end(a1)[7] ce qui signifie que l'utilisateur ne peut pas stopper a2 si a1 est actif.

7. il faut utiliser >, car sinon il s'agirait de `finish(a1, a2)`

objets STORM	d1	d2	δ1	δ2	Contraintes
seq (o1, o2)	F ,B	F ,B	F,B	F,B	begin(o2) $\geq$ end(o1)
seq_meet (o1, o2)	F,B	F,B	0	0	begin(o2)=end(o1)
seq_before (o1, o2)	F,B	F,B	F,B ($\neq$ 0)	F,B ($\neq$ 0)	begin(o2)=end(o1)+δ2
par (o1, o2)	F,B	F,B	F,B	F,B	
par_equal (o1, o2)	F,B	F,B	0	0	d1 = d2 $\wedge$ begin(o1)=begin(o2) $\wedge$ end(o2)=end(o2)
par_overlap (o1, o2)	F B B	F F B	F,B ($\neq$ 0)	F,B ($\neq$ 0)	d2 + δ2 > d1 $\wedge$ begin(o2)=begin(o1)+δ2 $\wedge$ end(o2)> end(o1) $\wedge$ begin(o2) < end(o1)
par_during (o1, o2)	F,B	F,B	F,B ($\neq$ 0)	F,B ($\neq$ 0)	d1 > d2 $\wedge$ begin(o2)=begin(o1)+δ2 $\wedge$ end(o2) < end(o1)
par_start (o1, o2)	F,B	F,B	0	0	begin(o2)=begin(o1)
par_finish (o1, o2)	F,B	F,B	F,B ($\neq$ 0)	F,B ($\neq$ 0)	d1 = d2 + δ2 $\wedge$ begin(o2)=begin(o1)+δ2 $\wedge$ end(o2)=end(o1)

TAB. 3.2 – *Cohérence des objets STORM.*

La Table 3.2 résume les présentations séquentielles et parallèles (en considérant seulement deux objets) avec durées, délais et contraintes associées. Dans cette table, la première colonne indique le type de la présentation. Les délais et les durées sont indiqués pour chaque opérande. La notation F,B signifie que les deux valeurs sont permises. Certaines contraintes semblent être redondantes mais il ne faut pas oublier que les durées peuvent être `free` ou `bound`. Par exemple, le qualifieur `equal` impose que les deux objets aient la même durée. Cependant, si les deux durées sont `free`, la contrainte statique n'est pas violée mais il est nécessaire, à la création, de démarrer les deux objets en même temps et, dès qu'on stoppe l'un des deux objets, l'autre doit être stoppé également.

3.6 Présentations intentionnelles

Cette partie présente une extension du modèle STORM original qui permet de spécifier des présentations de manière intentionnelle en utilisant un langage de requêtes (en l'occurrence le langage OQL) et des contraintes de synchronisation [Mar97]. L'avantage de cette approche est de pouvoir travailler sur une vision logique de l'information et donc de pouvoir présenter à chaque exécution, une présentation prenant en compte le contenu réel de la base de données. Comme dans le cas des vues dans les SGBD

relationnels, une requête permet de spécifier les critères de sélection de l'information à présenter.

On considère qu'on a une grande quantité d'informations stockée dans la base sous forme d'images, de sons, de textes ou de vidéos. Le but est de pouvoir présenter cette information à partir de recherches selon différents critères. Par exemple, présenter toutes les photos des membres du projet STORM. Pour cela, on effectue une recherche de photos dans la base dont un des associés est "membre du projet STORM". Nous associons alors des aspects temporels au résultat de la requête pour pouvoir le présenter à l'utilisateur de façon conviviale.

```
Class Employé                          Class Projet
tuple (Nom : string,                   tuple (intitulé : string,
       photo : Image,                         vidéo : Vidéo,
       projet : Projet,                       logo : Image,
       cv : Text)                             responsable : Employé)

Racines de Persistance
       name Les_Employés : list (Employé);
       name Les_Projets : set (Projet);
       name Logo_STORM : Image;
       name STORM : Projet;
       name Michel : Employé;
       name Hervé : Employé;
```

FIG. 3.4 – *Classes et Racines de Persistance*

Prenons comme exemple les deux requêtes suivantes `Q1` et `Q2` définies dans le langage standard OQL de l'ODMG [Cat93]. Elles sont basées sur les définitions données dans la figure 3.4.

Q1 : Sélectionner toutes les photos de tous les membres du projet STORM :

```
select e->photo
from   e in Les_Employés
where  e->projet->name = "STORM"
```

Cette requête peut faire partie d'une présentation intentionnelle qui présente les photos de tous les membres du projet STORM sous forme de diaporama à raison de 3 secondes par photo.

Q2 : Sélectionner les photos et CV des membres du projet STORM :

```
select tuple ( photo : e->photo, cv : e->cv)
from e in Les_Employés
where e->projet->name = "STORM"
```

Cette requête peut permettre de présenter à raison de 10 secondes par membre, les photos et CV des membres du projet STORM.

La requête sera exécutée au moment de jouer la présentation. Il est donc impossible de connaître à priori le nombre d'objets à présenter. Le comportement temporel de la requête est soit un comportement par défaut qui est calculé en fonction du type du résultat, soit le comportement spécifié au travers d'une contrainte de synchronisation. L'interprétation temporelle par défaut selon le type de données et le constructeur que nous utilisons est inspirée des résultats présentés dans [Adi96].

Afin de bénéficier des possibilités offertes par notre modèle (notamment pour pouvoir appliquer les méthodes correspondant à tout objet `SO`), le résultat de chacune de ces requêtes doit être un objet `SO` permettant d'intégrer ces présentations déclaratives de façon homogène au modèle STORM. Le contenu de l'objet est la représentation textuelle de la requête telle que définie précédemment. L'objet possède un délai qui vaut toujours zéro tandis que la durée de présentation est soit inhérente au type (cas des types audio et vidéo) soit `free` (c'est à dire que la fin de la présentation est soumise à une action utilisateur).

L'interprétation par défaut des constructeurs est :

- **list (Oi)**: les objets *Oi* sont présentés séquentiellement dans le même ordre que la liste.
- **set (Oi)**: les objets *Oi* sont présentés en parallèle.
- **tuple (O1, O2, ...,On)**: les objets *Oi* sont présentés en parallèle.

Chaque objet est présenté selon ses propres contraintes temporelles. Un type de données peut être une structure complexe. Par exemple, une liste peut être composée de n-uplets qui ont un attribut de type ensemble. Dans ce cas, l'interprétation par défaut est l'application récursive des règles énoncées ci-dessus.

Dans les exemples, en l'absence de contraintes, l'interprétation par défaut des requêtes est :

- **Q1**: une présentation séquentielle de toutes les photos (type du résultat = $list(Image)$).
- **Q2**: une présentation séquentielle de couples (photo, cv). Le contenu de chaque couple sera présenté en parallèle (type du résultat : $list(tuple(Image, Text))$).

L'interprétation inhérente au type et au constructeur peut cependant être remise en cause en définissant une contrainte de synchronisation temporelle sur la requête. En plus des contraintes usuelles sur les objets, nous introduisons les contraintes suivantes nécessaires à une interprétation ensembliste de la requête.

- $seq{-}before^n(C)$ spécifie que tous les objets de la collection doivent être présentés en séquence.
- $seq{-}meet^n(C)$ spécifie que tous les objets de la collection doivent être présentés en séquence sans délai entre deux présentations.
- $par{-}equal^n(C)$ spécifie que tous les objets de la collection doivent être présentés en parallèle. De plus, toutes les présentations doivent débuter et terminer en même temps. Cela équivaut à affecter la même ombre temporelle à tous les objets.
- $par{-}start^n(C)$ spécifie que tous les objets de la collection doivent être présentés en parallèle. De plus, toutes les présentations doivent débuter en même temps. Chaque objet peut posséder sa propre durée.
- $par{-}finish^n(C)$ spécifie que tous les objets de la collection doivent être présentés en parallèle. De plus, toutes les présentations doivent finir en même temps. Chaque objet peut posséder sa propre durée. La présentation termine dès que la présentation d'un objet est terminée.

Les contraintes *par-during* et *par-overlap* n'ont pas de sens si elles sont appliquées à une collection d'objets. Nous introduisons également une contrainte permettant de fragmenter la présentation d'une collection d'objets :

$block{-}size(C) = value$ spécifie le nombre d'éléments qui peuvent être présentés simultanément.

Dans les exemples définis, les contraintes à spécifier sont les suivantes :

- **Q1** : $seq{-}meet^{n}(select\ e \rightarrow photo\ from\ e\ in\ Les\_Employes\ where\ e \rightarrow projet \rightarrow name\ =\ \text{“}STORM\text{”})$;
- **Q2** : $seq-meet^{n}(select\ par-equal\ tuple(photo\ :\ e \rightarrow photo,\ cv\ :\ e \rightarrow cv)\ from\ e\ in\ Les\_Employes\ where\ e \rightarrow projet \rightarrow name\ =\ \text{“}STORM\text{”})$.

3.7 Exemples de présentations

Nous venons d'étudier comment sont modélisées les présentations multimédias dans le modèle STORM. Nous allons maintenant présenter comment elles sont créées et jouées à partir de différents exemples. Nous nous baserons sur le prototype que nous avons développé pour implanter les concepts de ce modèle [MMA96, Moc96a]. Ceci nous permet d'introduire l'indéterminisme dans le déroulement dans le temps d'une présentation lié aux aspects temporels des objets qui la composent.

3.7.1 Création d'une présentation

La création d'une présentation **d'un point de vue de l'utilisateur** se fait en deux étapes :

1. Créer ou rechercher les objets de la base qui feront partie de la présentation, et leur associer des aspects temporels et spatiaux. Ces objets peuvent être des images, des textes, des sons ou des vidéos.
2. Etablir la synchronisation parallèle ou séquentielle voulue entre ces objets.

Nous verrons au chapitre 5 comment la présentation est construite et stockée dans le SGBD par le gestionnaire de présentations en réponse aux différents choix de l'utilisateur. Ensuite, une fois créée, la présentation peut être jouée. Dans un premier temps, nous allons utiliser notre interface basée sur différents écrans et menus et, dans un deuxième temps, notre atelier de construction de présentations multimédias. Prenons l'exemple de la présentation multimédia “*Show_Grenoble*” dont le thème est la ville de Grenoble et dont le déroulement au cours du temps est présenté sur la figure 3.5. Il s'agit d'une présentation parallèle d'une musique, d'une séquence (Image, Vidéo) et d'un texte. Nous avons besoin de quatre objets monomédias pour créer cette présentation.

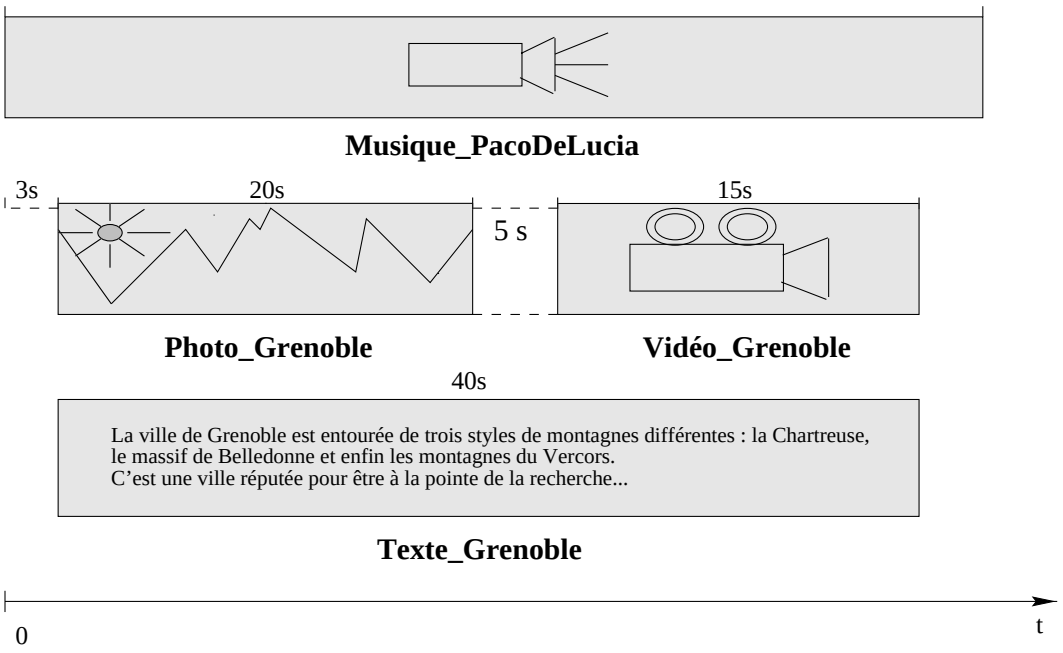

FIG. 3.5 – *Présentation de la ville de Grenoble*

La création de cette présentation, à l'aide de notre interface basée sur des écrans et des menus, recquiert à l'utilisateur de procéder aux étapes suivantes :

- Premièrement, il faut qu'il sélectionne les objets de la base qui seront présentés au moment de jouer la présentation. Si il existe déjà des images ou vidéos ou textes de Grenoble stockés dans la base, on effectue alors une recherche sur les objets qui ont pour sujet "Grenoble". L'utilisateur peut choisir les données monomédias à présenter qui lui conviennent ou en créer de nouvelles. L'image choisie, par exemple, a pour nom "Photo_Grenoble". Une fois l'image choisie, l'utilisateur définit les aspects temporels (délai et durée) et les aspects spatiaux (x et y qui sont les coordonnées dans l'espace pour les données affichables à l'écran) qui permettent de présenter cette image. L'utilisateur doit saisir ces données suivant le déroulement dans le temps et sur l'écran qu'il souhaite. La figure 3.6 présente les écrans pour saisir la présentation d'une image avec les apsects descriptifs d'abord (le nom, le sujet, les mot-clés, etc.), puis les aspects de présentation (les coordonnées, le délai et la durée).

  Remarque - L'image choisie fait partie d'un objet de la base, elle n'est pas dupliquée pour chaque présentation dont elle fait partie. Un objet `SO` est créé pour présenter l'image, il fait référence à travers son contenu à l'identificateur

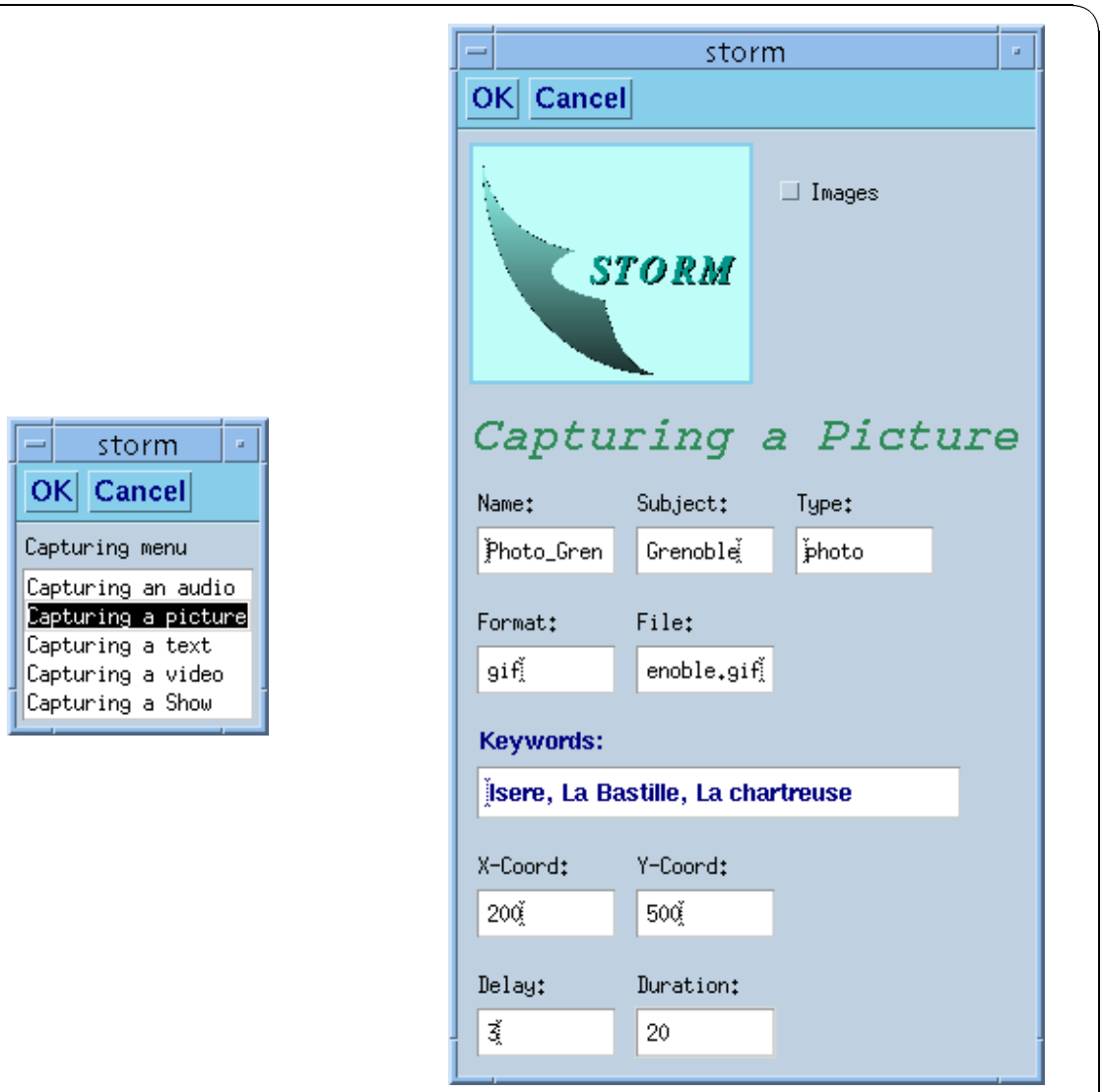

FIG. 3.6 – *Ecrans de saisie d'une image*

de cet objet image et il possède des aspects temporels et spatiaux. Ainsi l'image "Photo_Grenoble" possède plusieurs présentations sous forme de différents objets SO.

- Enfin, l'utilisateur doit définir la synchronisation entre les objets qu'il a choisis pour sa présentation. La figure 3.5 montre graphiquement le déroulement de la présentation au cours du temps et le type de synchronisation entre les différents objets. La synchronisation peut se décrire par l'expression suivante : *par_during(Musique_PacoDeLucia, par_equal(Texte_Grenoble, seq_before(Photo_Grenoble, Vidéo_Grenoble)))*.

Les objets `Photo_Grenoble` et `Vidéo_Grenoble` sont présentés en séquence, ils partagent la même ligne de temps. Au contraire, la synchronisation entre `Musique_PacoDeLucia`, `Texte_Grenoble` est `seq_before(Photo_Grenoble, Vidéo_Grenoble)`, ces objets possèdent ainsi leur propre ligne de temps.

L'utilisateur va définir trois contraintes de synchronisation : (1) la contrainte `seq_before` entre `Photo_Grenoble` et `Vidéo_Grenoble` impose un délai entre les présentations de ces objets; (2) la contrainte `par_equal` impose que la présentation du texte et de la séquence (Photo_Grenoble, Vidéo_Grenoble) commencent et se terminent en même temps; enfin (3) la synchronisation la plus générale est `par_during`, elle impose que la durée de la présentation de `Musique_PacoDeLucia` soit supérieure à la durée de la présentation de (`par_equal(Texte_Grenoble, seq_before(Photo_Grenoble, Vidéo_Grenoble)) + son délai`). La figure 3.7 présente l'écran pour le choix des opérateurs. L'utilisateur, après avoir saisi des objets `SO`, définit la synchronisation entre eux en choisissant une option de ce menu.

FIG. 3.7 – *Ecran pour le choix des opérateurs de synchronisation*

La présentation *Show_Grenoble*, une fois créée, est stockée sous forme d'un objet `SO`. Son contenu est complexe et décrit la synchronisation entre les objets `SO` qui le constituent. L'option "*play*" du menu permet de jouer la présentation en tenant compte du délai et de la durée des objets qui la composent et de la synchronisation entre ces objets. Il existe aussi les options "*edit*" et "*create a show*" qui permettent respectivement d'éditer l'objet `SO` correspondant à la présentation créée et de stocker cet objet dans la base. La figure 3.8 présente la vision sur l'écran de l'utilisateur lorsque sont présentés en même temps la photo de Grenoble, la vidéo et le texte.

Comme nous l'avons dit, nous pouvons aussi utiliser notre atelier de construction de présentations multimédias pour créer de manière graphique notre présentation

FIG. 3.8 – *Présentation à l'écran de la ville de Grenoble*

Show_Grenoble. L'annexe A présente en détails cet atelier. Le but de notre travail a été d'établir les bases d'un environnement de haut niveau pour la construction de présentations multimédias, facile à utiliser à l'aide de graphisme. Notre atelier est capable de créer toutes sortes de présentations cohérentes par rapport au modèle STORM. En effet, l'utilisation d'un atelier de conception pour la construction de telles présentations permet d'assurer que celles-ci respectent les règles de cohérence vis à vis du modèle de présentations.

La figure 3.9 montre la représentation graphique de la présentation *Show_Grenoble* dans l'atelier. Une présentation se construit à l'aide d'icones graphiques représentant les opérateurs de synchronisation ou les objets monomédias qui la composent. Premièrement, l'icone principal "*Show_Grenoble*" représente la présentation, si on clique deux fois apparaît un menu avec l'option "créer présentation" qui permet de créer le show une fois construit et de le jouer. Deuxièmement, trois icones de type "*opérateur*" représentent les contraintes de synchronisation entre les objets qui composent la présentation, de bas en haut, on trouve les opérateurs `par_during`, `par_equal` et `seq_before`. Enfin, les quatre autres icones représentent les objets `SO` qui composent la présentation. Ils appartiennent respectivement aux classes `SO_Audio`, `SO_Text`, `SO_Image` et

`SO_Video`. Si on clique deux fois sur ces objets, un menu apparaît avec une option "jouer", elle permet de jouer la présentation de l'objet monomédia. Une fois construite et validée par l'atelier, la présentation est aussi stockée dans la base sous forme d'un objet `SO`.

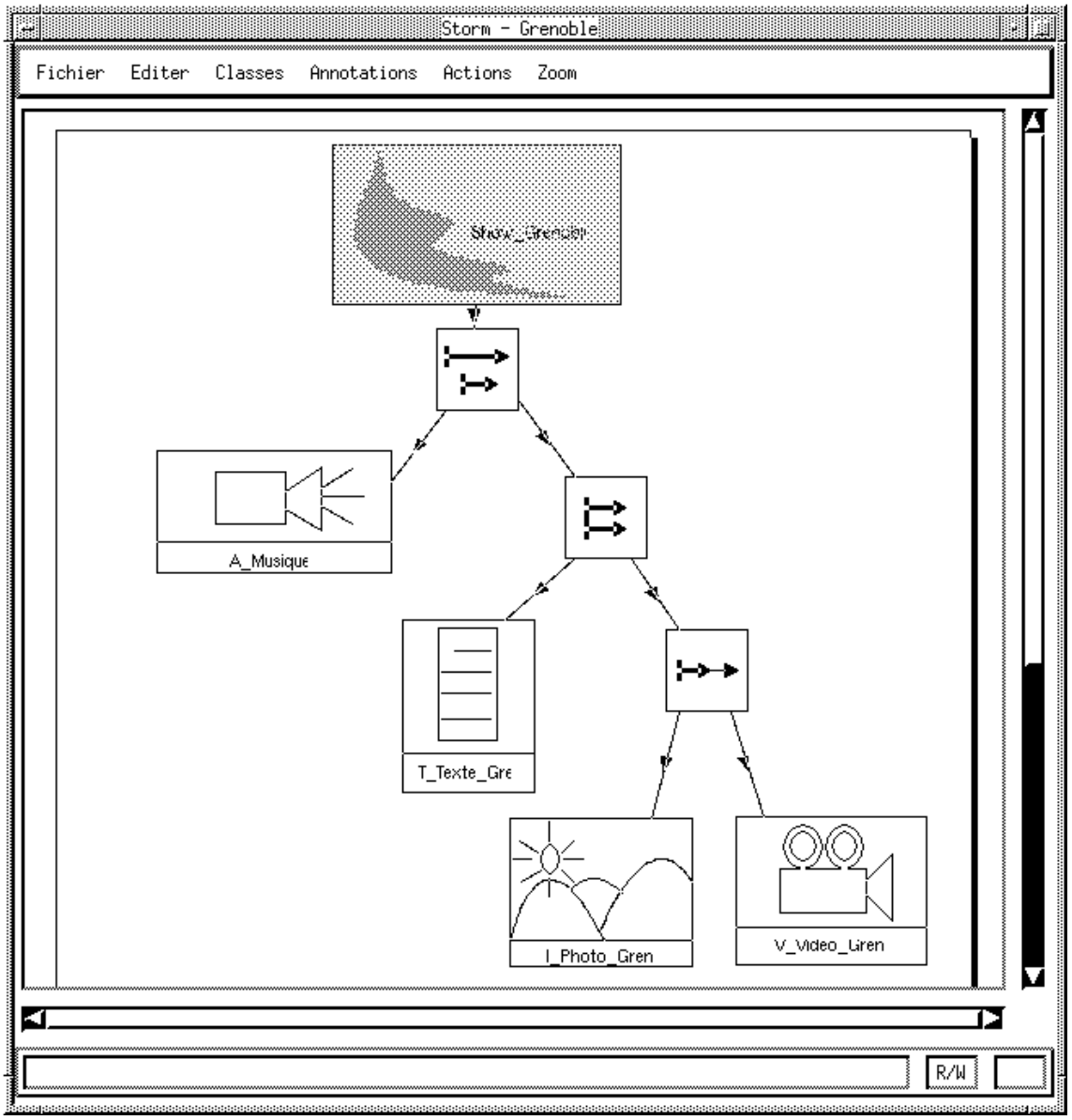

FIG. 3.9 – *Construction de la présentation de la ville de Grenoble*

Les aspects temporels liés aux objets de cette présentation sont tous `bound`. Ainsi, le déroulement de la présentation est déterminé et l'utilisateur ne peut pas interagir, il est un simple spectateur. Cependant, le modèle propose un certain indéterminisme

dans le déroulement temporel de la présentation si les aspects temporels ont une valeur `free` comme nous le montrons maintenant sur quelques exemples.

3.7.2 Indéterminisme dans une présentation

Si les aspects temporels prennent une valeur `free`, il apparaît un certain indéterminisme dans le déroulement d'une présentation. Si la **durée** de présentation d'un objet est `free`, c'est alors une action de l'utilisateur qui termine la présentation. Dans le cas d'un **délai** `free` pour la présentation d'un objet, l'observation de l'objet s'effectue seulement lorsque l'utilisateur le décide. Expliquons plus en détails ces deux notions :

- **Indéterminisme lié à la valeur illimitée d'un délai** :

  Dans le cas d'un délai illimité (`free`), nous affichons un icone ou un bouton montrant que le système est prêt à présenter l'objet mais attend qu'un événement se produise, typiquement une action de l'utilisateur.

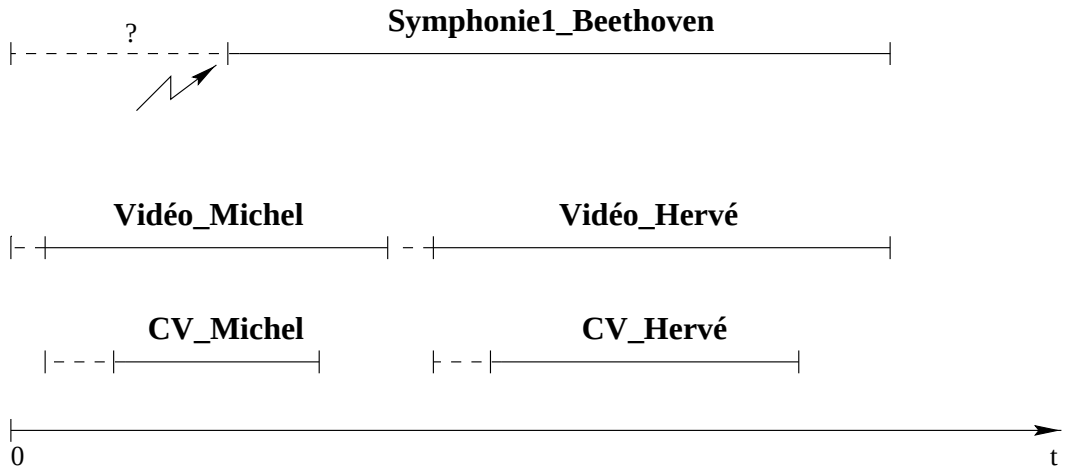

FIG. 3.10 – *Présentation avec déclenchement d'une musique de fond*

La figure 3.10 est une représentation graphique du déroulement dans le temps d'une présentation en parallèle d'une musique `M` et d'une séquence de couples (Vidéo, Texte) présentant les deux membres du projet STORM Michel et Hervé. La présentation est décrite par l'expression :
`par_finish(Symphonie1_Beethoven, seq_before(par_during(Vidéo_Michel, CV_Michel), par_during(Vidéo_Hervé, CV_Hervé))).`

La musique `Symphonie1_Beethoven` possède un délai illimité, ainsi l'utilisateur décide du moment où la musique démarre. Elle peut ne jamais être présentée si l'utilisateur n'interagit pas. Au début de la présentation est affiché un icone (ou bouton) montrant à l'utilisateur qu'il a la possibilité d'écouter une musique. Si l'utilisateur clique sur cet icone (événement déclencheur), la musique est jouée. Mais la contrainte `par_finish` impose qu'elle se termine en même temps que la séquence de couples (Vidéo, Texte).

L'exemple de la figure 3.11 montre une présentation séquentielle de couples (Image, Audio) présentant l'université d'accueil de l'équipe STORM et son batiment : `seq_before(par_finish(Photo_Université, Audio_Université), par_finish(Photo_BatimentD, Audio_BatimentD))`.

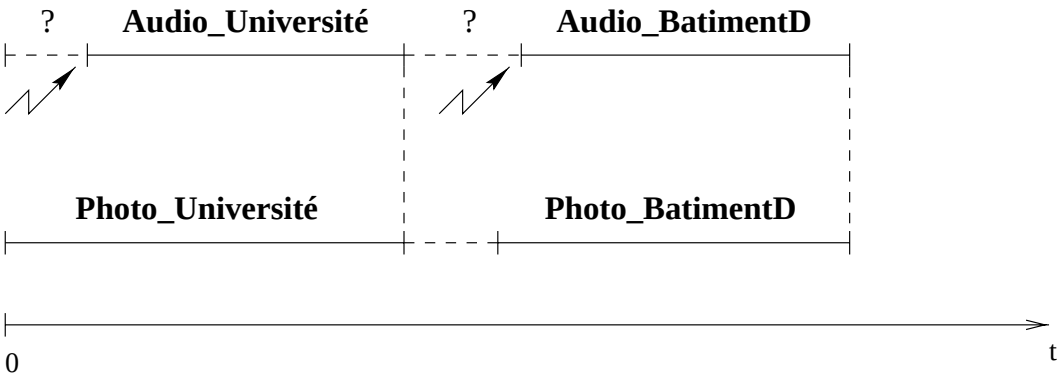

FIG. 3.11 – *Présentation séquentielle de couples (Image, Audio)*

Les objets audio `Audio_Université` et `Audio_BatimentD` possèdent un délai `free`. Par exemple, pour l'objet `Audio_Université`, un icone est affiché durant la durée d'affichage de l'image `Photo_Université`. L'utilisateur décide ou non d'écouter ce commentaire audio décrivant l'université. S'il veut l'entendre, il clique alors sur l'icone qui démarre `Audio_Université` et la contrainte de synchronisation `par_finish(Photo_Université, Audio_Université)` impose que les présentations `Photo_Université` et `Audio_Université` se terminent en même temps. Si l'utilisateur n'interagit pas pendant la présentation de `Photo_Université`, ensuite l'interaction n'est plus possible.

- **Indéterminisme lié à la valeur illimitée d'une durée** :

  Si la durée de présentation d'un objet est `free`, l'utilisateur choisit le moment où cette présentation se termine.

L'exemple de la figure 3.12 montre une présentation séquentielle d'images dont les durées sont `free`: `seq_meet(Photo_BatimentD, Photo_Bureau301, Photo_Bureau302, Photo_Bureau303)`. Les images sont affichées et seule une action de l'utilisateur peut stopper leur présentation pour pouvoir présenter la suivante.

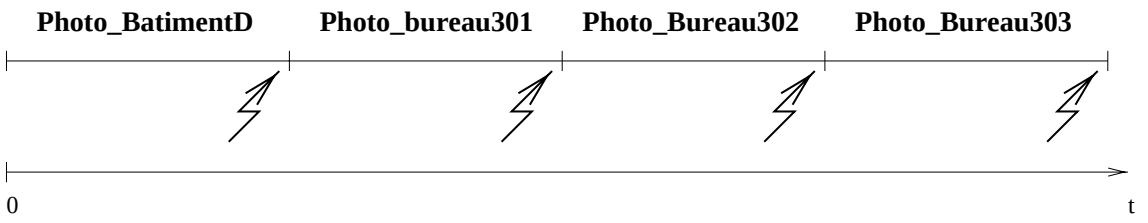

FIG. 3.12 – *Présentation séquentielle d'images avec des durées free*

Bilan - Nous pouvons constater que seuls les aspects temporels `free` entraînent un certain indéterminisme dans le déroulement temporel d'une présentation. Il s'agit de la seule interaction possible de l'utilisateur sur son déroulement. Nous allons chercher à augmenter le nombre d'interactions avec l'utilisateur et ainsi augmenter la dynamique des présentations.

3.8 Conclusion

Le modèle STORM propose une approche à objets pour les bases de données multimédias. Il permet avant tout la modélisation et la gestion des données multimédias telles que l'image, le texte, l'audio et la vidéo. Il propose ensuite des extensions aux SGBD à objets dans le but de construire, manipuler et jouer des présentations multimédias. Une présentation est composée de différents médias possèdant des aspects temporels et synchronisés entre eux à l'aide de différentes contraintes de synchronisation.

Les aspects temporels des objets sont définis par une *Ombre Temporelle* constituée d'un *délai* et d'une *durée* et les présentations sont stockées sous forme d'*objets STORM*. Ainsi, les présentations sont facilement recherchées, jouées et réutilisées. Le contenu d'un objet STORM est soit atomique pour la présentation d'un objet monomédia, soit composé pour la présentation de plusieurs objets. Le contenu peut aussi référencer une requête, l'objet est alors une présentation intentionnelle. La requête est exécutée au

moment de jouer la présentation et on ne connaît pas à priori le nombre d'objets à présenter. A partir des concepts du modèle, nous avons développé un prototype pour permettre à un utilisateur de construire, créer et jouer une présentation.

Les présentations créées suivant les concepts du modèle se déroulent suivant un scénario déterminé (un scénario étant un processus qui se déroule suivant un plan préétabli), leur déroulement dans le temps est prévisible. Nous proposons donc d'augmenter la dynamique des présentations en permettant d'exécuter différents comportements lors de leur déroulement. Pour cela, nous avons défini un *modèle de comportements* présenté au chapitre suivant.

Chapitre 4

Modèle de comportements d'une présentation multimédia

Comme nous l'avons vu au cours du chapitre précédent, le déroulement de nos présentations est prévisible et il dépend des aspects temporels associés à l'objet à présenter. La présentation d'un objet consiste à "jouer" individuellement un objet en fonction de son type (image, son, vidéo, texte) et de ses caractéristiques temporelles (modélisées par son *Ombre Temporelle*). Nous nous intéressons maintenant à la spécification de la dynamique d'une présentation multimédia.

Nous associons différents comportements à une présentation d'un objet. On appelle *comportement d'une présentation d'un objet*, l'ensemble des actions qu'il est susceptible d'entreprendre suite à des événements lors de sa présentation. En effet, le déroulement de la présentation d'un objet va être influencé par des événements de l'utilisateur ou par des événements provenant de la présentation d'autres objets. Ainsi on associe à un objet une *Ombre Temporelle* pour définir ses aspects temporels, et une *Ombre Comportementale* pour ses aspects comportementaux. Celle-ci joue deux rôles principaux : (1) changer le déroulement prédéfini de la présentation en un déroulement suivant différentes alternatives qui correspondent à différents comportements de la présentation et (2) segmenter la présentation en associant des actions à chaque segment. Notre modèle de comportements propose la modélisation de l'*Ombre Comportementale* qui est définie par un ensemble de comportements. Chaque comportement est modélisé par un type d'événement (déclencheur du comportement), un intervalle de validité et une liste d'actions à exécuter.

Notre modèle de comportements apporte seulement une solution au niveau de la

modélisation des comportements, nous proposons au chapitre suivant une implantation objet de ces comportements sous forme d'*objets scénarios* dans un but d'homogénéité de notre approche, mais aussi de réutilisabilité. Un certain nombre de comportements prédéfinis sont proposés au constructeur de la présentation. Cette modélisation objet nous permet aussi une interrogation des comportements à l'aide d'un langage d'interrogation comme `OQL`. Nous montrerons aussi comment utiliser la technologie des SGBD actifs pour exécuter nos présentations multimédias actives temporelles. Les règles actives (ou règles Evénement-condition-Action) permettent d'exécuter les différents comportements liés aux objets de la présentation.

La Section 4.1 définit la notion d'*Ombre Comportementale* associée à un objet `SO`. La Section 4.2 propose la définition de l'*Ombre Comportementale*. Les Sections suivantes présentent les applications possibles de l'*Ombre Comportementale*. Elles correspondent soit à des extensions temporelles, soit à différents comportements que l'on peut trouver au cours d'une présentation. Enfin, la Section 4.7 propose une classification des Ombres Comportementales.

4.1 L'Ombre Comportementale

Dans le modèle de présentations multimédias, une présentation inclut : (1) les objets à présenter, (2) les contraintes temporelles à leur appliquer (délai, durée) et (3) les contraintes de synchronisation et de transition entre objets (l'un après l'autre, en parallèle, etc.). Une présentation est stockée sous forme d'un objet STORM (ou `SO`), défini par un quadruplet (i, δ, d, c) où `i` est un identificateur. Le contenu `c` décrit l'objet monomédia à présenter ou la composition avec des contraintes de synchronisation de différents objets à présenter.

Le modèle de présentations est centré sur la notion d'*Ombre Temporelle* qui définit les valeurs temporelles associées à un objet dans le temps. En effet, une présentation a un déroulement prévisible et prédéfini dans le temps. La liberté d'exécution est insuffisante, il faut pouvoir interagir avec la présentation, modifier son comportement par défaut.

Le but du modèle de comportements des présentations est de rajouter une nouvelle dimension aux objets STORM pour pouvoir leur associer un comportement particulier au cours de leur présentation. L'ensemble des comportements lié à la présentation constitue l'*Ombre Comportementale* des présentations. Le déroulement de la présentation devient imprévisible et dépend des événements qui se produisent et qui déclenchent

une suite d'actions. Ces réactions à différents événements augmentent la dynamique de l'exécution de la présentation. Les événements peuvent provenir d'actions de l'utilisateur ou alors du comportement d'autres objets.

Un objet STORM correspondant à une présentation d'un objet monomédia possède une *Ombre Comportementale* par défaut nulle. Aucune action de l'utilisateur n'est possible et il n'existe aucune influence provenant d'autres objets de la présentation. Par contre, s'il fait partie d'une présentation plus complexe, tout objet STORM possède une *Ombre Comportementale* par défaut. En effet, si on prend le déroulement de la présentation `par_overlap(o1, o2)` (c.f. figure 4.1), l'objet `SO o1` a une *Ombre Comportementale* par défaut qui définit qu'au bout de 10 s après le début de `o1`, on démarre `o2`. La contrainte de synchronisation `par_overlap` impose aussi un comportement aux deux objets `SO o1` et `o2` : le début de `o2` entraîne le début de l'observation de `o1` et la fin de `o2` entraîne la fin de `o1` (le paragraphe 4.3.2 décrit plus en détails les comportements liés aux différentes contraintes de synchronisation).

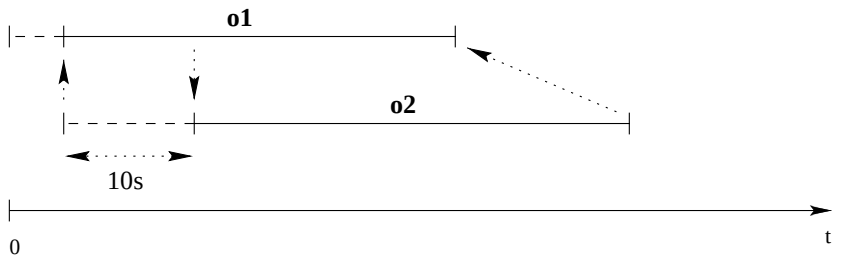

FIG. 4.1 – *Présentation par_overlap(o1, o2)*

Une *Ombre Comportementale* a deux rôles principaux, chacun d'eux montrant les changements qu'elle peut entraîner sur le déroulement de la présentation, il s'agit de :

- *changer le déroulement prédéfini de la présentation en un déroulement suivant différentes alternatives* qui correspondent à différents comportements de la présentation. Le déroulement de la présentation peut être dévié de son déroulement dans le temps prédéfini au départ suite à un événement. Par exemple, durant un intervalle de la présentation, on propose à l'utilisateur différents choix. Suivant ce choix, le scénario de la présentation varie. Sur l'exemple de la figure 4.2, lorsque se déroule la présentation d'une photo du massif de Belledonne, on propose à l'utilisateur de visionner une vidéo de cette chaîne de montagne à l'aide d'un bouton `bt` qui est présent durant un intervalle au cours de cette présentation. Si l'utilisateur clique sur le bouton, on stoppe la présentation en cours et on lance

la vidéo de Belledonne.

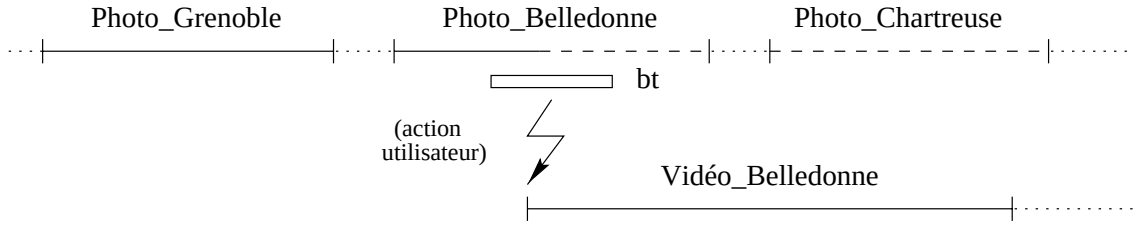

FIG. 4.2 – *Présentation des alentours de la ville de Grenoble*

- *segmenter la présentation en associant des actions à chaque segment.* Sur l'exemple de la figure 4.3, on segmente la présentation de la vidéo de Belledonne. A chaque segment est associé un comportement particulier et il s'exécute seulement sur un intervalle de temps au cours de cette présentation. Au cours du segment "`invisible`", la vidéo n'est plus visible à l'écran même si elle continue à se dérouler. Au cours du segment "`ralenti`", la vidéo est présentée au ralenti. Et enfin, au cours du segment "`stop`" l'utilisateur a la possibilité d'arrêter la présentation.

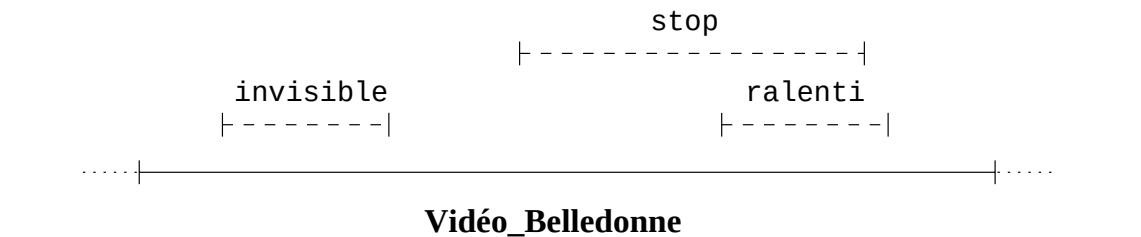

FIG. 4.3 – *Segmentation d'une présentation d'une vidéo*

Lorsque l'on segmente une présentation, il apparaît clairement certaines vérifications à effectuer pour qu'elle soit cohérente. Il faut vérifier la compatibilité des actions associées aux différents segments. Il est impossible d'associer à une présentation deux segments de type "invisible" et "ralenti" dont les intervalles dans le temps se chevauchent. L'exécution de la présentation devient incohérente.

4.2 Définition de l'Ombre Comportementale

L'*Ombre Comportementale* définit les comportements liés à une présentation. Le comportement est déclenché par un événement et exécute une liste d'actions. Ce comportement peut ne pas se produire si l'événement qui le déclenche n'a pas lieu au cours de la présentation. Ainsi apparaît de l'indéterminisme dans le déroulement d'une présentation dépendant des événements externes ou internes qui se produisent.

Définition -

Un objet `SO` est défini dorénavant comme un **quintuplet (i, δ, d, c, b)** où **b** constitue l'*Ombre Comportementale*. `b` correspond à une liste de tuple constitué d'un intervalle de validité **iv**, d'un type d'événements **te** et d'une liste d'actions **a**. Chaque tuple définit un comportement particulier associé à l'objet. La réalisation de ce comportement peut ne pas se produire au cours de la présentation de l'objet `SO` puisqu'il dépend de l'arrivée d'un événement (surtout si il s'agit d'une action de l'utilisateur).

Soit un objet **SO** *défini par le quintuplet (i, δ, d, c, b),*
son Ombre Comportementale **b** *est définie par :*

b = *list(tuple(E: te_i, I: iv_i, A: a_i))*

avec $te_{i:1..n}$ *: un type d'événements et*
$iv_{i:1..n}$ *: un intervalle de validité et*
$a_{i:1..n}$ *: une liste d'actions*

FIG. 4.4 – *Un objet SO*

Définition -

L'*Ombre Comportementale* est définie par une **liste de tuple(I: iv_i, E: te_i, A: a_i)** avec i:1..n. **te**$_i$ représente le type de l'événement déclencheur du comportement, **iv**$_i$ l'intervalle de validité de l'événement de type `te`$_i$, **a**$_i$ correspond à la liste d'actions à exécuter suite à une occurrence de `te`$_i$, c'est à dire un événement.

Le type de l'événement déclencheur te -

Le type d'événements **te** détermine le type de comportement déclenchable. Les différents types d'événements que l'on manipule sont groupés en deux grandes catégories

d'événements : les types d'événements externes $\mathbf{te}^e$ et internes $\mathbf{te}^i$.

$\mathbf{te} : \{\ te^e,\ te^i\ \}$ avec $te^e : \{\ te^u,\ te^n,\ te^m\ \}$ et $te^i : \{\ te^a,\ te^t\ \}$

où te^u : *type d'événements utilisateurs,*
te^n : *type d'événements de navigation,*
te^m : *type d'événements de manipulation,*
te^a : *type d'événements d'appel de méthodes,*
te^t : *type d'événements temporels.*

FIG. 4.5 – *Les types d'événements*

Nos différents types d'événements nous permettent de classifier les différents types de comportements qui peuvent être observés au cours d'une présentation, ils sont présentés à la fin de ce chapitre dans le tableau 4.3. Le comportement associé au type d'événements est seulement déclenché si cet événement se produit durant son intervalle de validité.

L'intervalle de validité iv -

Un événement de type **te** peut se produire seulement au cours d'un intervalle nommé intervalle de validité **iv**. L'intervalle de validité par défaut correspond à un intervalle dont les bornes sont respectivement le temps de début et le temps de fin de la présentation ([0, δ + d]). Autrement, les bornes de l'intervalle défini doivent faire partie de l'intervalle [0, δ + d].

$\mathbf{iv} = [\ b_{inf},\ b_{sup}\ ]$ avec b_{inf} : *un entier* ≥ 0, b_{sup} : *un entier* > 0,
$0 \leq b_{inf} \leq \delta + d$,
$0 \leq b_{sup} \leq \delta + d$

Par défaut, $\mathbf{iv} = [0, \delta + d]$
Si $b_{inf} = b_{sup}$ *alors* $\mathbf{iv}$ = *instant de validité*

FIG. 4.6 – *L'intervalle de validité*

Un événement peut aussi n'être valide qu'à un instant du temps, on parle alors d'**instant de validité**. Dans ce cas, les bornes de l'intervalle de validité sont égales. Dans la définition d'un intervalle de validité, on retrouve la notion de segmentation

de la présentation. En effet, un intervalle de validité définit un intervalle de temps au cours de la présentation où un événement spécifique peut se produire. Et cet événement déclenche une action particulière. Ainsi, différents segments sont associés à la présentation et ils sont liés à une action.

La liste des actions a -

a représente la liste des actions à exécuter suite à un événement de type **te**. Ce sont elles qui réalisent le déroulement du scénario.

$$\mathbf{a} = list(a_1, a_2, .., a_n) \qquad avec\ a_{i:1..n} : une\ action$$

FIG. 4.7 – *Les actions*

Les actions sont associées soit à l'objet `SO` lié à ce comportement, soit aux objets cibles qui participent au comportement. Les actions correspondent, dans de nombreux cas, à des appels de méthodes sur ces objets.

Nous allons maintenant nous intéresser aux différents comportements que l'on peut associer à une présentation et comment ils sont pris en compte au niveau de l'*Ombre Comportementale*.

4.3 Ombre Comportementale pour les aspects temporels

Une présentation multimédia synchronise des objets multimédias possèdant des aspects temporels (un *délai* et *une durée*). Lorsque la valeur de ces aspects temporels devient variable, ce déroulement devient imprévisible. Par exemple, si on donne à l'utilisateur la possibilité d'interagir avec la présentation, il n'est pas possible de prévoir le moment exact où aura lieu l'action de l'utilisateur. Le déroulement de la présentation varie suivant l'action de l'utilisateur. Nous avons défini, au paragraphe 3.7.2, l'indéterminisme lié aux aspects temporels proposé dans le modèle STORM. Si le délai ou la durée d'une présentation prennent une valeur `free`, seule une action de l'utilisateur détermine leur fin.

Nous proposons des extensions temporelles permettant d'augmenter la dynamique d'une présentation. L'ajout d'indéterminisme dans le déroulement temporel d'une pré-

sentation va entraîner des problèmes pour la vérification des contraintes de synchronisation, ainsi nous proposons une nouvelle façon de les vérifier en utilisant des relations de causalité.

4.3.1 Délai et durée interactifs

Les aspects temporels associés à un objet multimédia sont : un *délai* et une *durée*. Ils peuvent prendre soit une valeur `bound` (limitée), soit une valeur `free` (illimitée). Ces concepts ont été présentés au chapitre 3.

Lorsqu'une *durée* ou un *délai* prend une valeur `bound`, l'utilisateur n'a pas la possibilité d'interrompre la présentation, il est passif par rapport à son déroulement. Nous proposons une combinaison des deux valeurs `bound` et `free` pour étendre le type d'interactivité possible. Il s'agit d'interactions limitées dans le temps.

En effet, nous pourrons ainsi donner une valeur "`bound`" à la durée de présentation d'un objet, mais définir un intervalle de temps où l'utilisateur peut interagir (pour arrêter la présentation). Par exemple, on associe une durée de 1 mn à la présentation d'une image (celle-ci est alors présentée pendant 1 mn au maximum), mais on définit qu'à partir de la 30ème seconde d'affichage de l'image, l'utilisateur peut interagir pour l'effacer. De même, pour le délai d'une présentation, par exemple, si un délai a une durée `bound` de 30 secondes, on peut tout de même donner la possibilité à l'utilisateur de démarrer plus tôt la présentation de l'objet (à partir de la 15ème seconde par exemple) en lui permettant d'interagir.

Ainsi on peut associer respectivement au délai et à la durée d'une présentation deux nouveaux aspects temporels interactifs : un **délai interactif** et une **durée interactive**. Ils sont définissables seulement si la valeur du délai ou de la durée associée est `bound`. La figure 4.8 présente un exemple de ces nouveaux aspects temporels. δ et `d` correspondent au délai et à la durée, alors que δ_i représente le *délai interactif* et d_i la *durée interactive*.

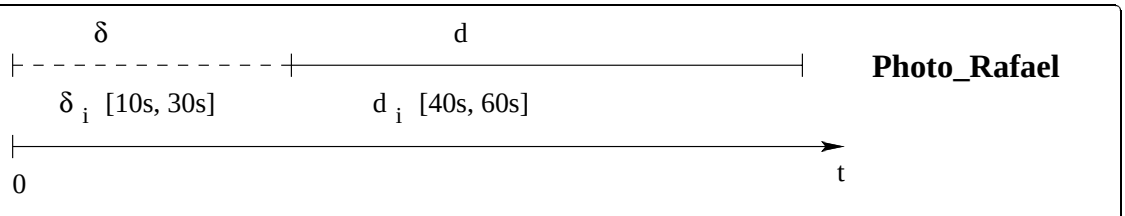

FIG. 4.8 – *Représentation d'un délai interactif et d'une durée interactive*

- **Exemple de délai interactif**

  La figure 4.9 présente une séquence entre une musique et une vidéo: `seq( Musique_Mozart, Vidéo_STORM)`.

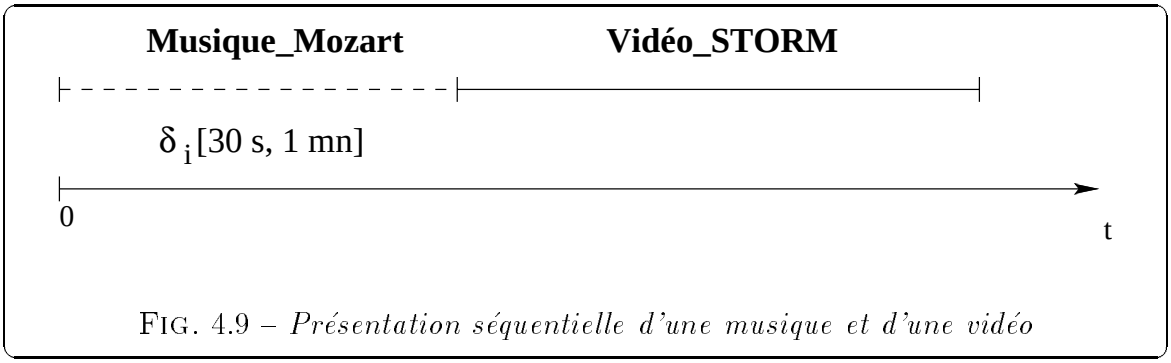

FIG. 4.9 – *Présentation séquentielle d'une musique et d'une vidéo*

Si le délai de `Musique_Mozart` est `free`, la musique commence seulement si l'utilisateur interagit (clic sur l'icone montrant que la musique est prête à être écoutée). Or si aucune contrainte de synchronisation n'impose la fin de la présentation de `Musique_Mozart`, alors la vidéo n'est pas présentée. Nous décidons de limiter l'interaction dans le temps pour qu'elle s'arrête même si l'utilisateur n'a pas interagi et que la présentation de `Vidéo_STORM` commence, par exemple au bout d'une minute.

Pour définir cette possibilité, le délai de `Musique_Mozart` prend une valeur `bound` égale à 1 mn et nous lui associons une valeur pour le **délai interactif** égale à [30s, 1mn]. La borne inférieure représente le moment où l'interaction devient possible (30 secondes après le début de la présentation de `Musique_Mozart`), c'est à dire que si un événement utilisateur se produit, il est pris en compte et la présentation de la musique `Musique_Mozart` débute. Au bout d'une minute d'attente, si aucune action de l'utilisateur n'a eu lieu, l'interaction n'est plus possible, elle s'arrête et on commence la présentation de `Vidéo_STORM`.

- **Exemple de durée interactive**

  Dans le cas où on veut afficher une image pendant un certain temps (correspondant à sa durée), mais donner à l'utilisateur la possibilité de l'effacer avant cette limite, on associe alors une durée interactive à sa présentation. Par exemple, une image `Photo_doctorants` à présenter possède une **durée interactive** égale à [10s, 30s] (voir figure 4.10). Cette valeur définit qu'au bout de dix secondes de présentation l'utilisateur peut interargir. En effet, entre dix secondes et trente secondes de présentation, l'utilisateur a la possibilité d'effacer l'image. S'il reste

passif, la fin de la présentation de `Photo_doctorants` a lieu au bout de quarante secondes.

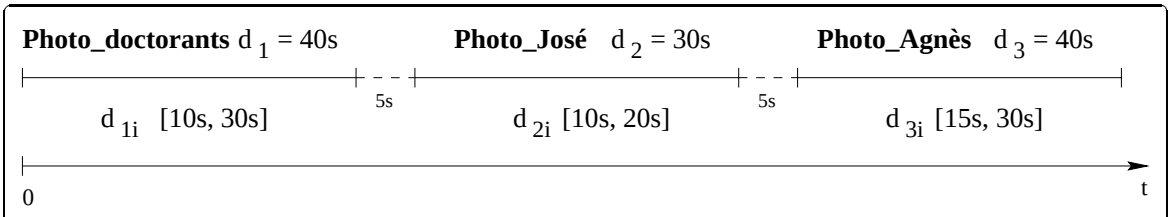

FIG. 4.10 – *Présentation séquentielle d'images*

En conclusion, dans le cas d'une valeur `bound`, on associe aux présentations deux autres aspects temporels : un **délai interactif** et une **durée interactive**. Ils sont représentés par un intervalle de temps qui définit le temps de validité où une action de l'utilisateur peut se produire.

délai/durée	délai/durée interactif
F ($[0, +\infty[$)	------
B ($[0, d]$ ou $[0, \delta]$)	$[\alpha, \beta]$ où $\alpha \geq 0$ et $\beta \leq d$ (ou δ)

TAB. 4.1 – *Correspondance entre les valeurs des aspects temporels*

- **Prise en compte au niveau de l'Ombre Comportementale**

Un délai et une durée interactive sont associés à un objet `SO` et ils définissent un comportement particulier. Ils permettent à l'utilisateur d'interagir sur la présentation pour la démarrer ou la stopper. Ces deux comportements font partie de l'*Ombre Comportementale* de l'objet `SO`.

Exemple -

Soit l'objet $\mathbf{so}_1 = (i_1, \delta_1, d_1, c_1, b_1) \in$ `SO` *avec* $\delta_1 = 10s$ *et* $d_1 = 30s$,
son Ombre Comportementale $\mathbf{b}_1$ *est définie par :*

$\mathbf{b}_1 = list(tuple(E: te^u_1, I: iv_1, A: a_1), tuple(E: te^u_2, I: iv_2, A: a_2))$ *où*

- *le premier tuple correspond au* **délai interactif** *avec*

te^u_1 : *type d'événements utilisateur,*
iv_1 = *[5, 10],* /* $iv_1 = \delta_i$ */
a_1 = *list*(a_{11}) *avec* a_{11} : *démarrer* so_1.

- *le deuxième tuple correspond à la* **durée interactive** *avec*
  te^u_2 : *type d'événements utilisateur,*
  iv_2 = *[20, 30],* /* $iv_1 = d_i$ */
  a_2 = *list*(a_{21}) *avec* a_{21} : *stopper* so_1.

Ces deux comportements sont déclenchés par un type d'événements utilisateurs. L'intervalle de validité de l'événement va correspondre à l'intervalle que l'on souhaite associer au délai interactif ou à la durée interactive. Enfin, dans les deux cas, une seule action est exécutée suite à l'événement utilisateur, soit on démarre la présentation, soit on la stoppe.

4.3.2 Les relations de causalité

L'ajout d'interactions dans les aspects temporels implique de l'indéterminisme dans le déroulement dans le temps de la présentation d'un objet multimédia et entraîne des problèmes dans la vérification des contraintes de synchronisation entre les objets qui composent la présentation. En effet, les valeurs temporelles des durées et des délais varient suivant l'action de l'utilisateur. Nous ne pouvons plus vérifier le respect des contraintes à partir de ces valeurs.

Nous introduisons des relations de causalité dans la sémantique de nos contraintes de synchronisation. Elles définissent la manière de procéder pour que les contraintes soient respectées. Les incohérences sont alors systématiquement éliminées. Une relation de causalité entre deux objets impose des dépendances causales entre le *début* et *la fin* de la présentation des objets. Par exemple, la fin de la présentation d'un objet entraîne le début de la présentation d'un autre objet.

La figure 4.11 présente les relations temporelles possibles entre les objets multimédias. Les flèches représentent les dépendances causales entre les objets. Le sens de la flèche indique que telle action (comme le démarrage ou la fin de la présentation d'un objet) entraîne telle autre action. Les objets `so1` et `so2` sont deux objets `SO` : (i1, δ1, d1, c1, b1) et (i2, δ2, d2, c2, b2).

Dans le cas de la contrainte `par_equal(so1, so2)`, les présentations de `so1` et `so2` démarrent et finissent en même temps, leurs délais étant nuls. Il existe quatre

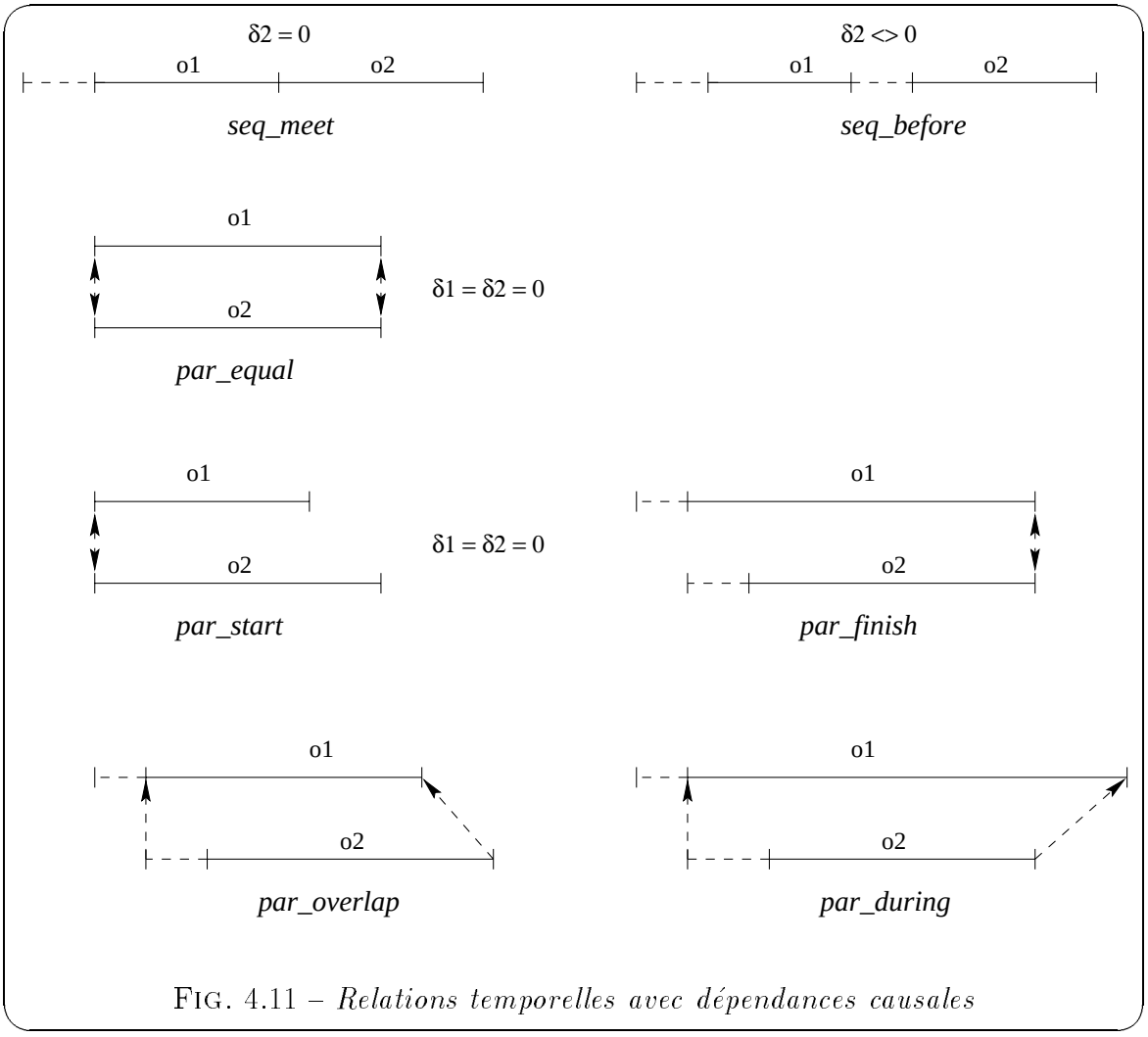

FIG. 4.11 – *Relations temporelles avec dépendances causales*

dépendances causales entre le début et la fin de la présentation des deux objets. Premièrement, le début de la présentation de `so1` démarre la présentation de `so2` et inversement. Deuxièmement, la fin de la présentation de `so1` entraîne la fin de la présentation de `so2` et inversement.

Ces comportements liés à la contrainte `par_equal` sont définis au niveau des *Ombres Comportementales* `b1` de `so1` et `b2` de `so2`. Par exemple, `b1` est définie de la façon suivante :

$\mathbf{b}_1$ = *list(tuple(E:* te^a_1*, I:* iv_1*, A:* a_1*), tuple(E:* te^a_2*, I:* iv_2*, A:* a_2*)) où*

- *le premier tuple correspond à la contrainte : "le début de so1 démarre so2" avec*
  te^a_1 *: type d'événements d'appel de méthodes /* début de so1 */,*
  iv_1 *= [0, 0],*

a_1 = *list*(a_{11}) *avec* a_{11} : *démarrer* so_2.

- *le deuxième tuple correspond à la contrainte : "la fin de so1 stoppe so2" avec*
  te^a_2 : *type d'événements d'appel de méthodes* /* *fin de so1* */,
  $iv_2 = [\delta + d, \delta + d]$,
  a_2 = *list*(a_{21}) *avec* a_{21} : *stopper* so_2.

Les comportements permettant de réaliser les relations de causalité sont déclenchés par des événements d'appel de méthodes. Dans ce cas-là, le comportement d'un objet agit sur le comportement des autres objets auquel il est lié par une contrainte de synchronisation. Les intervalles de validité sont ici des instants de validité qui correspondent au début et à la fin de la présentation.

Le tableau 4.2 nous donne textuellement la sémantique des relations temporelles auxquelles se sont ajoutées les dépendances causales entre le début et la fin des présentations des objets.

4.4 Ombre Comportementale pour un objet d'une présentation

Nous nous intéressons tout d'abord aux comportements liés à un objet `SO` correspondant à la présentation d'un objet monomédia et faisant partie d'une présentation plus complexe. Ces comportements sont de deux types : (1) *lien navigationnel vers une autre présentation* ou (2) *contrainte de synchronisation spécifique*.

4.4.1 Lien navigationnel vers une autre présentation

Un lien navigationnel va permettre à un utilisateur de passer d'une présentation à une autre. Un lien est établi sur un objet de la présentation principale et aboutit à une autre présentation.

Ce concept est identique à l'"*hypermédia*" qui permet de naviguer d'une page de données à une autre suivant des liens établis. Le déclenchement de cette autre présentation peut entraîner l'arrêt ou seulement une pause de la présentation principale.

Nous allons maintenant donner un exemple d'un lien déclenchant une autre présentation en séquence. Dans notre application présentant le projet STORM, nous présen-

Contrainte de synchronisation	Délais	Sémantique de la contrainte
seq_meet (so1, so2)	δ_2=0	La fin de la présentation de l'objet so1 entraîne le début de la présentation de l'objet so2 sans délai entre elles.
seq_before (so1, so2)	δ_2<>0	La fin de la présentation de l'objet so1 entraîne le début de la présentation de l'objet so2 sans délai entre elles.
par_equal (so1, so2)	δ_1=δ_2=0	Les présentations des objets so1 et so2 débutent en même temps et elles sont stoppées lorsque le premier des deux intervalles est terminé. L'égalité est toujours vérifiée même si, initialement, les deux intervalles n'avaient pas la même durée.
par_overlap (so1, so2)		Le début de la présentation de so2 entraîne le début de la présentation de so1 si celle-ci ne s'est pas produite avant. De même, la fin de la présentation de so2 entraîne la fin de so1 si elle n'a pas eu lieu.
par_during (so1, so2)		Le début de la présentation de so2 entraîne le début de la présentation de so1 si celle-ci ne s'est pas produite avant. De même, la fin de la présentation de so1 entraîne la fin de so2 si elle n'a pas eu lieu.
par_start (so1, so2)	δ_1=δ_2=0	Les présentations des objets so1 et so2 démarrent en même temps.
par_finish (so1, so2)		La première présentation qui finit entraîne la fin de la deuxième.

TAB. 4.2 – *Les relations temporelles*

tons une séquence décrivant les membres du projet avec en parallèle une musique de fond, la présentation d'un membre est une présentation parallèle de sa photo et d'un texte correspondant à son Curriculum Vitæ. Si l'utilisateur s'intéresse plus particulièrement à un des membres, nous lui proposons de cliquer sur sa photo et ainsi d'obtenir une présentation d'une vidéo de la personne avec un commentaire audio en parallèle. Le constructeur de la présentation décide, par exemple, que le début de cette autre présentation `Show_Agnès` entraîne la fin de la présentation principale `Show_Membres`.

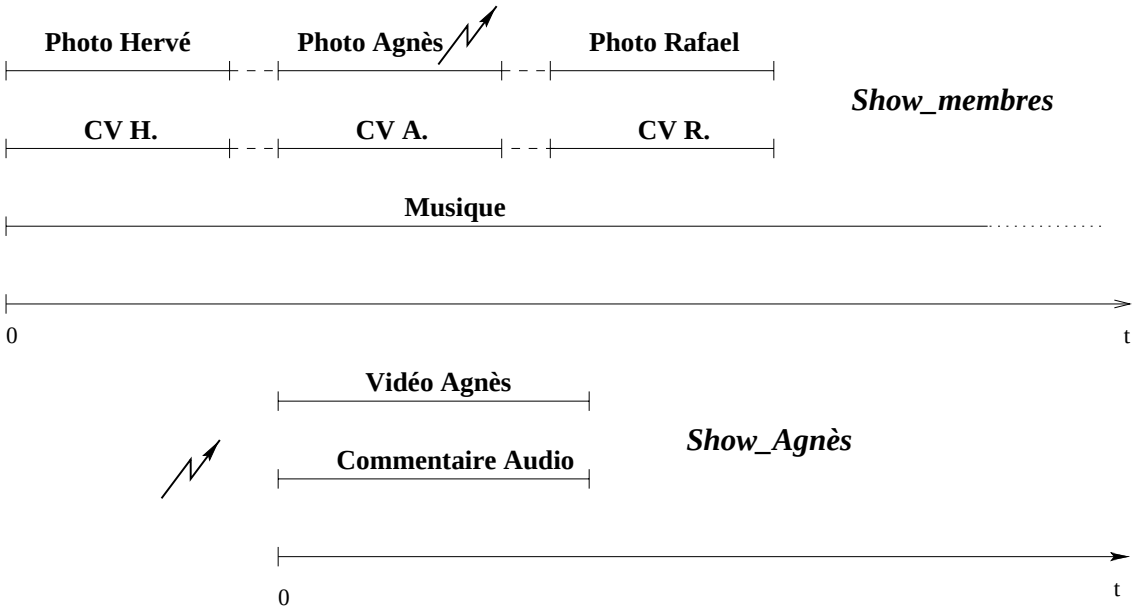

FIG. 4.12 – *Lien navigationnel vers une autre présentation*

Mais il aurait pu choisir de faire seulement une pause de `Show_Membres`, ainsi lorsque `Show_Agnès` se termine, `Show_Membres` redémarre. La figure 4.12 nous donne la représentation graphique du déroulement d'une telle présentation.

L'*Ombre Comportementale* `b` de l'objet `Photo Agnès` définit ce comportement :

b = *list(tuple(E: te^n, I: iv, A: a)) où*

- *le tuple correspond au* **lien navigationnel**,
  te^n : type d'événements navigationnel,
  iv = [0, δ + d],
  a = *list*(a$_1$, a$_2$) *avec* a$_1$ *: stopper Show_Membres,*
  a$_2$ *: démarrer Show_Agnès.*

L'intervalle de validité est l'intervalle par défaut, c'est à dire qu'il correspond à l'intervalle de présentation de `Photo Agnès`, mais il pourrait être plus limité dans le temps.

4.4.2 Contrainte de synchronisation spécifique

Certaines contraintes de synchronisation ne sont pas exprimables à l'aide du modèle de présentations multimédias. Nous allons le prouver sur l'exemple de la figure 4.13. Cette présentation démarre avec une vidéo et un commentaire audio d'introduction. En parallèle, démarre la présentation d'un texte expliquant les thèmes de recherche abordés dans le projet, ainsi qu'un commentaire audio du chef de projet et sa photo. A la fin, une vidéo est présentée montrant les membres du projet. Trois contraintes de synchronisation apparaissent : (1) une contrainte `par_equal` entre les objets `Vidéo_Intro` et `Audio_Intro`, (2) une contrainte `par_finish` entre `Texte_PrésentationProjet` et `Image_Chef` et enfin, (3) une contrainte `par_finish` entre `Vidéo_Membres` et `Audio_Chef`.

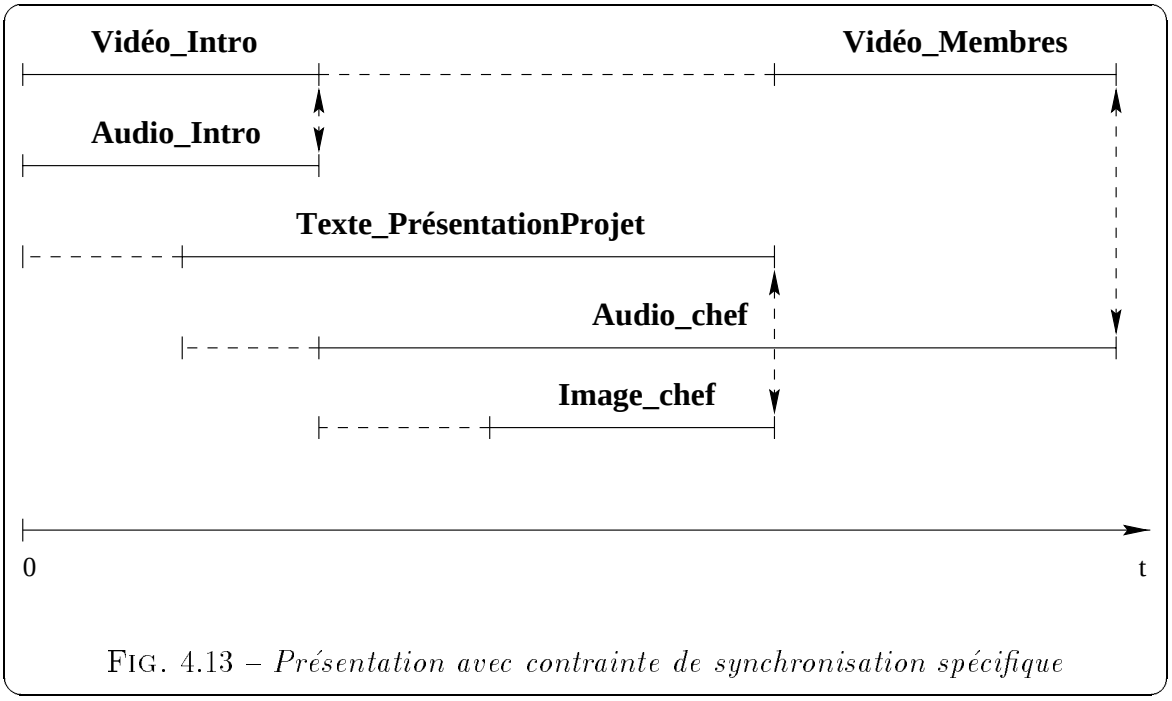

FIG. 4.13 – *Présentation avec contrainte de synchronisation spécifique*

Le déroulement de la présentation multimédia décrite sur cette figure peut aussi être exprimé à l'aide de l'expression suivante : *par(par_finish(par(seq(par_equal(Vidéo_Intro, Audio_Intro), Vidéo_Membres), Texte_PresentationProjet), Audio_Chef), Image_Chef)* pour respecter la contrainte de synchronisation `par_finish` entre les objets `Vidéo_Membres` et `Audio_Chef`.

Par contre, la contrainte `par_finish` entre les objets `Texte_PresentationProjet` et `Image_Chef` n'est pas exprimée, il n'y a pas de combinaison possible pour respecter les deux contraintes à la fois. On va alors associer un comportement spécifique à ces

objets qui permet d'exprimer cette contrainte de synchronisation, c'est à dire que les deux objets doivent se terminer en même temps.

Les contraintes de synchronisation spécifiques d'une présentation sont définies au niveau de l'*Ombre Comportementale*. Leur comportement est identique au comportement des relations de causalité que nous avons détaillé au paragraphe 4.3.2.

L'intégration de contraintes de synchronisation spécifiques dans une présentation nous permet d'augmenter la souplesse dans la définition des contraintes de synchronisation de la présentation. Les contraintes de synchronisation seront définies suivant les besoins de l'utilisateur.

4.5 Ombre Comportementale pour une présentation

Une *Ombre Comportementale* peut aussi être rattachée à une présentation complexe composée de différents objets. On définit des comportements globaux à cette présentation.

4.5.1 Comportement lié à la manipulation au cours du temps de la présentation

Différentes actions influent un comportement au déroulement d'une présentation au cours du temps comme un arrêt, une pause, un retour arrière, une avance rapide, etc. Ces actions peuvent être appliquées à toute présentation multimédia à l'aide de boutons affichés au cours de la présentation, chacun d'eux ayant un rôle précis (`STOP`, `PAUSE`, `PLAY`, `REWIND`, `FORWARD`). Ce type de comportement ne sera pas modélisé, il est du domaine de l'interface. Par contre, nous offrons des méthodes pour réaliser ces différentes opérations.

Le bouton `STOP` permet d'arrêter la présentation, sans redémarrage possible. Le bouton `PAUSE` permet de stopper momentanément la présentation, et l'utilisateur peut la redémarrer au même endroit en utilisant le bouton `PLAY`. Enfin, les boutons `REWIND` et `FORWARD` permettent de revenir en arrière ou d'avancer dans le déroulement de la présentation.

4.5.2 Comportement lié à des événements temporels

Différents comportements peuvent aussi être observés au cours de la présentation suite à un événement temporel. On définit, par exemple, qu'au bout de 20 secondes après le début de la présentation, on démarre la présentation d'un autre objet, comme une musique.

Le comportement est alors inclu dans l'*Ombre Comportementale* de la présentation :

$\mathbf{b}_p$ = *list(tuple(E:* $te^t{}_p$*, I:* iv_p*, A:* a_p*))* *où*

$te^t{}_p$ *: événement temporel,*
iv_p = *[20, 20],*
a_p = *list(*a_{p1}*) avec* a_{p1} *: démarrer le show "Musique".*

Dans le cas des événements temporels, l'intervalle de validité qui leur est associé est un instant de validité correspondant à l'instant du temps où ils se produisent.

4.6 Ombre Comportementale pour une requête d'une présentation

Le comportement d'une requête d'une présentation dépend soit d'*événements temporels*, soit d'*appels de méthodes.*

Prenons un exemple de requête pouvant faire partie d'une présentation et étudions les différents comportements qui peuvent se produire. La notion de requête comme objet composant d'une présentation est explicitée au chapitre 3. La requête suivante `R1` peut être rattachée à un objet `SO` et permettrait, par exemple, de présenter, à raison de 10 secondes, les photos et Curriculum Vitae des membres du projet STORM :

R1 : select *tuple(photo: e->photo, cv: e->cv)*
from *e in Les_Employés*
where *e->projet->intitulé = "STORM"*

Le résultat de la requête est une liste, il s'agit donc d'une présentation séquentielle. Par définition de la sémantique du constructeur `tuple`, la photo et le cv seront présentés en parallèle. La figure 4.14 nous montre le déroulement dans le temps de la

présentation de cette requête. Au cours de la présentation de cette requête, il peut se produire différents événements qui déclenchent un comportement spécifique à chacun.

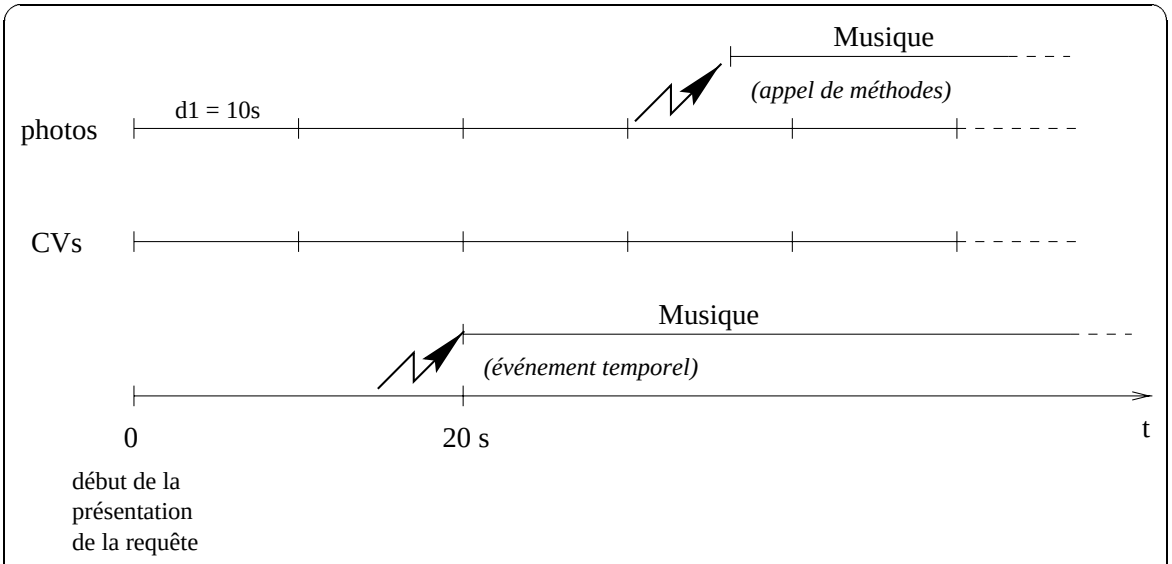

FIG. 4.14 – *Présentation d'une requête*

4.6.1 Comportement lié à des événements temporels

Différents comportements peuvent être déclenchés par un événement temporel au cours de la présentation d'une collection. On peut définir, par exemple, qu'au bout de 20 secondes après le début de la présentation de la requête, on déclenche la présentation d'un autre objet, comme une musique. La figure 4.14 nous présente un exemple de ce type d'événement.

Suivant le type de comportement que l'on veut réaliser, les actions déclenchées varient et peuvent être :

- lancement de la présentation d'un objet `SO`,
- changement des paramètres de présentation (la durée de la présentation de chaque couple (photo, CV) est modifiée).

4.6.2 Comportement lié à des appels de méthodes

Un autre type de comportement de la collection peut être lié à un événement d'appel de méthodes. Nous exprimons, par exemple, qu'une certaine action est réalisée lorsque le quatrième événement d'appel de la méthode `start_presentation` se produit après le début de la présentation de la requête. L'action définit le type de comportement que l'on veut déclencher au cours de la présentation de la requête. Sur l'exemple de la figure 4.14, une musique démarre suite à cet appel de méthodes.

Soit l'objet $\mathbf{so}_q$ *présentant le résultat de la requête* `R1`,

son Ombre Comportementale $\mathbf{b}_q$ *est définie ainsi :*
$\mathbf{b}_q$ = *list(tuple(E:* te^t_{q1}, *I:* iv_{q1}, *A:* a_{q1}*), tuple(E:* te^a_{q2}, *I:* iv_{q2}, *A:* a_{q2}*)) où*

- *le premier tuple correspond au* **comportement temporel**,
  te^t_{q1} *: type d'événements temporel,*
  iv_{q1} = *[20, 20],*
  a_{q1} = *list(*a_{q11}*) avec* a_{q11} *: démarrer le show "Musique".*

- *le deuxième tuple correspond au* **comportement lié à un appel de méthode**,
  te^a_{q2} *: type d'événements d'appel de méthodes,*
  iv_{q2} = *[0,* δ + *d],*
  a_{q2} = *list(*a_{q21}*) avec* a_{q21} *: démarrer le show "Musique".*

4.7 Classification des Ombres Comportementales

Chaque comportement est réalisé suite à un événement. La table 4.3 nous présente tous les types de comportements qui peuvent être observés au cours d'une présentation. Un type de comportement est décrit par le type d'événements qui est le déclencheur du comportement, par l'intervalle de validité du type d'événements et les actions déclenchées pour réaliser le comportement.

Nous distinguons deux grandes catégories d'événements susceptibles de déclencher un comportement, les **événements externes** et les **événements internes** :

Catégories des événements	**Types des événements**	**Intervalle de validité par défaut**	**Actions**
Evénements externes	• événement utilisateur	iv = δ_i (délai interactif) iv = d_i (durée interactive)	• exécution de méthodes : - play (délai `F` ou `I`) - end_presentation (durée `F` ou `I`)
	• événement de navigation	[0, δ + d]	• démarrer un nouveau show
	• événement de manipulation	[0, δ + d]	• changement du déroulement dans le temps de la présentation (start, stop, pause, resume, etc.)
Evénements internes	• événement d'appel de méthodes : - début présentation - début observation - fin présentation - début de la présentation d'une requête	[0, δ + d]	• exécution de méthodes
	• événement temporel	[v_t, v_t] v_t : instant du temps où l'événement survient	• exécution de méthodes : - lancement d'un autre objet - changement des paramètres de présentation d'une requête
	• événement système (problèmes liés aux ressources)	[0, δ + d]	• arrêt de la présentation

TAB. 4.3 – *Les comportements dans une présentation multimédia*

1. Les *événements externes* au SGBD sont relatifs à des événements résultant d'une

action de l'utilisateur. Ils sont classifiés en trois types d'événements :

- Les *événements utilisateurs* déclenchent les comportements liés aux aspects temporels interactifs associés à la présentation d'un objet multimédia. L'objet concerné est un objet `SO` qui possède un délai interactif ou une durée interactive. Ces événements sont déclenchés par une action de l'utilisateur.

  Le comportement lié à ce type d'événements entraîne soit le démarrage de l'observation de l'objet pour un délai `free` ou `interactif` , soit l'arrêt de la présentation pour une durée `free` ou `interactive`. Aucun autre objet ne participe à ce comportement.

- Les *événements de navigation* sont déclenchables au cours de la présentation d'un objet, ils sont déclenchés par une action de l'utilisateur qui indique à celui-ci qu'il peut visionner une autre présentation multimédia relative à la présentation en cours.

  Le comportement lié à ce type d'événements réalise le comportement permettant de naviguer d'une présentation à une autre suivant un lien navigationnel. L'action exécutée suite à l'événement est le démarrage d'un nouveau show, celui-ci correspond à un objet cible qui participe au comportement.

- Les *événements de manipulation* sont liés à une présentation (un objet `SO` concerné) et sont déclenchés par une action de l'utilisateur sur un des boutons "stop", "pause", "resume", etc. Les différentes actions déclenchées par ce type d'événement effectuent différents changements sur le déroulement de la présentation : arrêt de la présentation (méthode `stop` de l'objet `SO`), pause sur la présentation (méthode `pause`), reprise de la présentation (méthode `resume`), etc.

  Remarque - On ne modélisera pas les comportements liés aux *événements de manipulation*, en effet ils seront gérés par l'interface et il n'est pas necessaire de stocker leur comportement dans la base, ceux-ci étant prédéfinis et ne variant pas. Le SGBD doit seulement fournir des méthodes pour gérer les pauses dans une présentation, les reprises, les retours arrières, etc.

2. Les *événements internes* au SGBD résultant d'actions liées à des objets comme les méthodes (*événements d'appel de méthodes*), mais aussi liées au temps (*événements temporels*).

   - Les *événements d'appel de méthodes* correspondent à des appels de méthodes sur des objets `SO` (`play`, `start_presentation`, etc.).

Les comportements déclenchés par ce type d'événements sont soit liés à une requête d'une présentation, soit liés à des objets d'une présentation pour exécuter des contraintes de synchronisation spécifiques. Les actions déclenchées par ces événements sont des méthodes sur d'autres objets `SO`.

- Les *événements temporels* sont déclenchés à un temps donné relatif à l'exécution d'une action particulière. Cette action correspond à l'appel d'une méthode sur l'objet `SO` concerné.

  Les comportements exécutés par ce type d'événements sont liés soit à une requête d'une présentation, soit à une présentation. Les actions déclenchées sont des exécutions de méthodes sur d'autres objets `SO`.

- Les *événements systèmes* sont déclenchés par les ressources du système. Ils indiquent en général un problème dans son fonctionnement. Et ils peuvent entraîner l'arrêt de la présentation.

  Remarque - On ne modélisera pas non plus les *événements systèmes*, mais les ressources produisent des messages qui peuvent déclencher des réactions sur le déroulement d'une présentation.

4.8 Conclusion

Une présentation multimédia a un caractère dynamique inhérent dû aux données multimédias présentées. Notre *modèle de comportements* propose d'augmenter ce dynamisme en associant une *Ombre Comportementale* à la présentation. Une *Ombre Comportementale* définit un ensemble de comportements associés à la présentation d'un objet. Un comportement est capable d'entreprendre un ensemble d'actions suite à un événement lors de sa présentation. Chaque comportement est décrit par un type d'événements, un intervalle de validité et une liste d'actions.

Nous avons défini différents comportements pouvant être déclenchés au cours de la présentation et nous avons étudié comment ils pouvaient être exprimés à l'aide de l'*Ombre Comportementale*. Les aspects temporels spécifiés dans le modèle STORM ont nécessité certaines extensions pour permettre une plus grande expression et une plus grande liberté au niveau de l'interaction avec l'utilisateur. Deux notions d'intervalles ont été ajoutées : délai interactif et durée interactive. Ils permettent de définir deux comportements spécifiques définissant que l'utilisateur a la possibilité d'interagir avec la présentation pour soit démarrer la présentation, soit la stopper. Enfin, nous avons classifié les différents événements pouvant se produire au cours d'une présentation dans

le but de spécifier tous les types de comportements possibles.

Nous décrivons, dans le chapitre suivant, notre choix dans l'implantation de l'*Ombre Comportementale* sous forme d'*objets scénarios*. Les deux principaux avantages de cette démarche sont : (1) la réutilisabilité des comportements dans une application (ou entre plusieurs applications); (2) et l'interrogation des différents comportements à l'aide d'un langage de requêtes. Ensuite, nous présentons notre approche proposant l'*exécution des comportements à l'aide de règles actives*. En effet, la sémantique des *objets scénario* est très proche de celle d'une règle active Evénement-Condition-Action (*lorsqu'un événement se produit et si la condition est satisfaite alors exécuter l'action*). Ainsi, ces objets sont traduits en règles actives pour pouvoir être joués au cours de la présentation.

L'originalité de notre approche par rapport aux autres travaux comme [HFK95, AL96, VS96, Vaz96] est notre volonté de ne pas se limiter à la définition d'*interactions avec l'utilisateur*, mais d'élargir cette définition aux comportements d'une présentation en réponse à différents événements (utilisateurs ou pas). Seule l'approche MHEG [Pri93, Sa94b, MBE95] parle de *comportements*, le terme "comportement" englobe toutes les activités concernant la présentation des objets au niveau de l'interface, l'interaction avec l'utilisateur est définie à part. Mais ces activités se limitent aux aspects temporels et spatiaux, ainsi qu'aux relations entre les objets `rt-objects` (*runtime objects*, ces objets correspondant à nos objets `SO`).

Chapitre 5

Réalisation d'un SGBD multimédia

Le modèle de présentations multimédias offre différents concepts permettant d'intégrer, de manipuler et de rechercher des données multimédias dans un SGBD, ainsi que la construction et l'interrogation de présentations multimédias. Le modèle de comportements offre des concepts permettant d'associer différents comportements à une présentation, pour que son déroulement devienne imprévisible. En nous basant sur ces deux modèles, nous montrons comment étendre un SGBD à objets conventionnel pour offrir des fonctionnalités d'aide à la construction d'applications multimédias.

Il faut tout d'abord décrire l'architecture fonctionnelle du SGBD multimédia. L'intégration et la manipulation des données multimédias doivent se faire à tous les niveaux de l'architecture. L'interface doit posséder des outils permettant de présenter à l'utilisateur les objets multimédias de la base. Le gestionnaire de présentations se charge de construire les différentes présentations en synchronisant différents objets entre eux, il permet aussi de les présenter en respect de leur structure et enfin de les interroger une fois qu'elles ont été stockées dans la base. Enfin, le serveur d'objets comprend le gestionnaire d'objets qui permet de stocker les objets dans la base de données. Notre réalisation s'est faite au-dessus du SGBD O_2. Il réalise notre définition et conception d'un SGBD multimédia, il montre ainsi la faisabilité des concepts énoncés au cours de cette thèse [MMA96, Moc96a, Moc96b, ALMM97].

Les extensions multimédias proposées sont réalisées à l'aide d'une **bibliothèque de classes**. Les instances de ces classes sont des objets STORM possédant des aspects temporels et leur contenu fait référence à des objets multimédias de la base (de type image, texte, vidéo, audio, etc.). Cette bibliothèque peut être utilisée dans chaque nouvelle application et peut être étendue suivant les besoins de l'utilisateur. Nous

décrirons ensuite comment utiliser ces fonctionnalités multimédias pour construire des applications. Nous verrons comment construire des présentations multimédias, les jouer et les interroger. Pour l'interrogation, nous utilisons le langage standard OQL et nous montrons les extensions nécessaires pour élargir les possibilités de recherche des données mutlimédias.

Enfin, nous proposons notre implantation des comportements sous forme d'une **bibliothèque de classes de comportements**. Les instances de ces classes sont des *objets scénarios*. Ensuite, nous décrivons notre choix dans l'exécution des comportements et le respect des contraintes de synchronisation à l'aide de *règles actives*. En effet, la sémantique des comportements liés à un objet d'une présentation est très proche de celle d'une règle active Evénement-Condition-Action (*lorsqu'un événement se produit et si la condition est satisfaite alors exécuter l'action*). Ainsi, les objets scénarios sont traduits en règles actives pour pouvoir être joués au cours de la présentation. Nous allons ainsi utiliser la technologie des SGBD actifs pour développer notre SGBD multimédia. Plus particulièrement, nous nous appuierons sur le système actif NAOS qui permet la définition et l'exécution de règles actives pour les applications O_2 [CCS94]. NAOS a, en effet, été développé en considérant le SGBD O_2 comme noyau de gestion des données et d'exécution des applications. Il s'intègre dans l'architecture modulaire de ce système et constitue donc un nouveau composant utilisable pour le développement des applications, à côté des autres composants élémentaires que sont O_2C, l'interface C++ et O_2SQL.

La Section 5.1 présente l'architecture fonctionnelle de notre SGBD multimédia avec son interface utilisateur, son gestionnaire de présentations et son serveur d'objets. La Section 5.2 décrit les différentes extensions multimédias au SGBD à objets proposées : (1) les différents objets multimédias qui permettent de manipuler et stocker les données multimédias et (2) la bibliothèque de classes STORM dont les instances sont des présentations des objets de la base. La Section 5.3 décrit comment construire les présentations multimédias, les jouer et les interroger. Enfin, la Section 5.4 présente la gestion des comportements d'une présentation, elle décrit notre bibliothèque de comportements, l'interrogation des comportements, leur exécution et leur création.

5.1 Architecture

Au lieu de redéfinir un nouveau SGBD, nous proposons d'étendre les SGBD à objets existants et d'utiliser, si possible, leurs outils pour gérer les objets mutlimédias.

Nous utilisons un modèle de données objet pour modéliser la structure spécifique de chaque média et modéliser les présentations multimédias.

L'architecture fonctionnelle du SGBD multimédia se compose de trois niveaux, de bas en haut : (1) le serveur d'objets comprenant le gestionnaire d'objets qui stocke les objets et les valeurs, (2) le gestionnaire de présentations, (3) l'interface utilisateur.

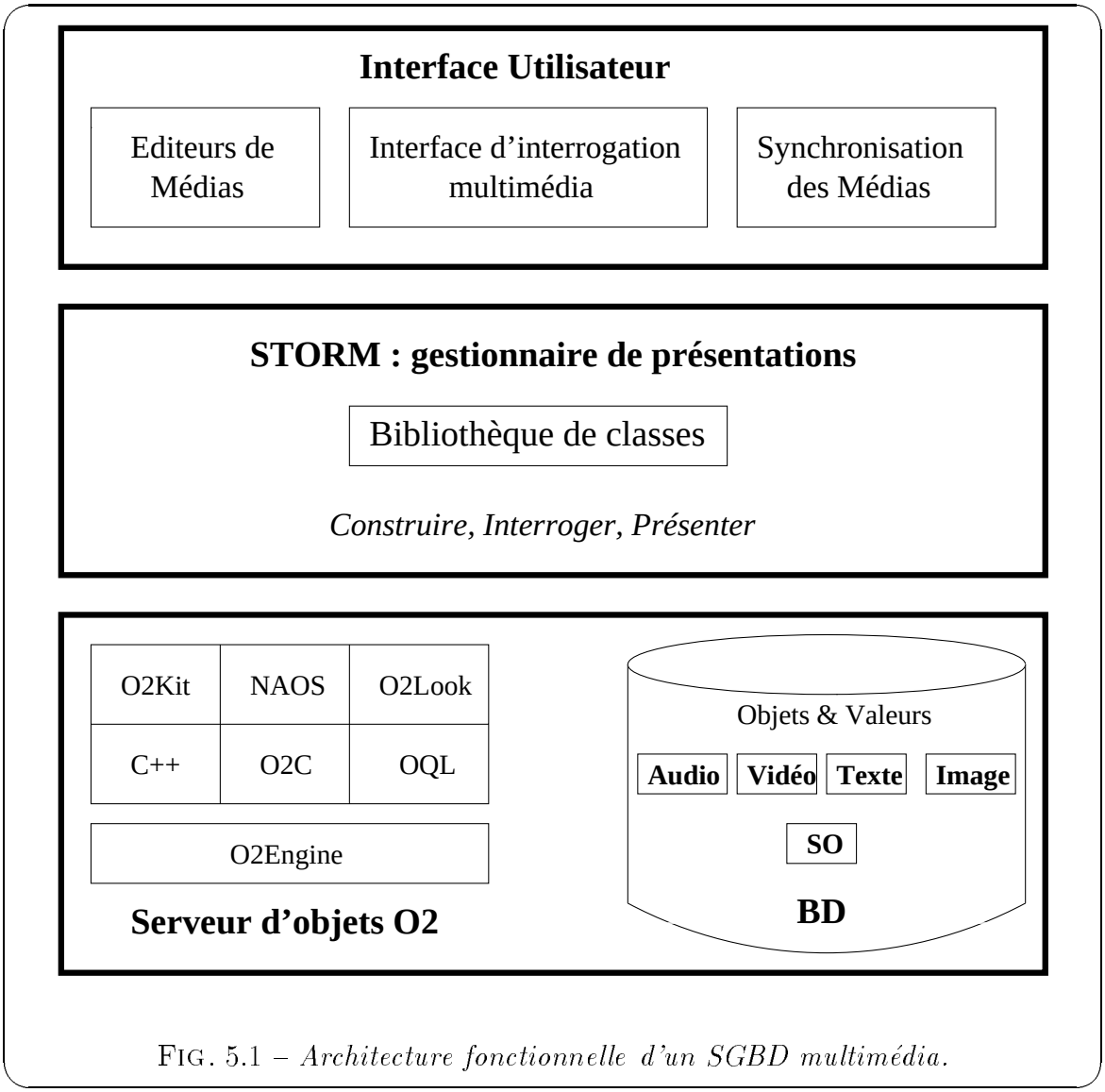

FIG. 5.1 – *Architecture fonctionnelle d'un SGBD multimédia.*

5.1.1 Le serveur d'objets

Le *serveur d'objets* prend en charge les aspects classiques d'un SGBD : gestion d'un schéma, stockage d'objets, requêtes, transactions, concurrence, reprise. Mais il doit

maintenant offrir des possibilités pour gérer chaque type de média avec des techniques de stockage adaptées à chacun.

Il doit aussi offrir une représentation canonique de chaque type de données monomédias pour pouvoir saisir leur structure complexe. A l'heure actuelle, de nombreux SGBD multimédias relationnels comme *Sybase*, *UniSQL* ou orienté-objet comme O_2, *ObjectStore* supportent les BLOB (Binary Large Object) pour les données multimédias. Un BLOB est non typé, long et avec un champ de longueur variable utilisé pour stocker des données multimédias dans la base. Le SGBD stocke la donnée multimédia comme une donnée non typée et délivre simplement des blocs de données à l'application qui la demande.

Pour réaliser ce prototype, le serveur d'objets utilisé est le SGBD O_2 [BDK92, Tec93, AC93]. Le système O_2 est un Système de Gestion de Bases de Données pouvant être interfacé à des langages de programmation standards comme C++. Il possède un environnement de programmation complet composé d'un langage de quatrième génération O_2C, d'un langage de requêtes OQL (conforme au standard ODMG) et d'une interface graphique O_2Look. A la base de O_2, on trouve le moteur O_2Engine, qui présente toutes les caractéristiques d'un système de Bases de données et celles d'un système orienté-objet.

5.1.2 Le gestionnaire de présentations multimédias

Le *gestionnaire de présentations multimédias* modélise les relations entre les objets pour pouvoir créer, rechercher et jouer des présentations multimédias complexes. Elles sont composées de différents objets avec des contraintes spatiales, temporelles et comportementales.

Nous proposons des extensions multimédias sous forme de **bibliothèques de classes** (que nous décrivons au paragraphe 5.2). Nous retrouvons le concept d'*Ombre Temporelle* qui consiste à ajouter des propriétés temporelles (délai et durée) à chaque objet apparaissant dans une présentation. Nous utilisons le langage OQL qui est un standard de l'ODMG [Cat93] pour interroger nos présentations multimédias. Nous proposons certaines extensions à ce langage pour permettre de les interroger sur leurs attributs temporels ou sur la synchronisation entre les objets qui les composent.

Enfin, la troisième fonctionnalité du gestionnaire est de pouvoir gérer les comportements liés à la présentation. Nous présenterons, au cours du paragraphe 5.4.1, la bibliothèque de classes de comportements proposée et ensuite la façon dont les com-

portements peuvent être exécutés.

Notre gestionnaire de présentations est une extension du SGBD O_2. Il a été développé en C++ (à l'aide de l'interface C++ proposée par O_2) pour des raisons techniques et dans une optique de portabilité vis à vis du système O_2. L'avantage de développer nos extensions multimédias en une application C++ est que nous avons pu utiliser des processus légers et des sémaphores. Le mécanisme de processus légers nous permet de gérer le parallélisme entre les objets d'une présentation présentés en parallèle.

5.1.3 L'interface utilisateur

L'*interface utilisateur* est composée d'outils pour l'édition des données multimédias. Elle offre des services pour l'affichage d'images, pour jouer l'audio et un outil de visualisation de vidéos. Elle fournit des outils pour créer et jouer les présentations multimédias en accord avec leurs contraintes de synchronisation.

Enfin, elle fournit des services pour l'interrogation des données multimédias et des présentations multimédias. La plus simple requête est une interrogation pour accéder et délivrer une donnée multimédia spécifique. Des recherches plus complexes offrent aux applications la capacité d'effectuer des recherches basées à la fois sur les attributs et sur les relations de synchronisation des données multimédias. Par exemple, rechercher toutes les vidéos avec pour mot-clés "projet STORM" et "LSR-IMAG" ou toutes les vidéos du projet STORM relatives à l'audio "MyWay". Cette dernière requête combine la connaissance du SGBD sur la synchronisation entre l'audio et la vidéo.

5.1.3.1 Développement en O_2Look

O_2Look est une interface graphique utilisateur pour le SGBD O_2. Elle opère audessus du système X-Window, utilise le gestionnaire graphique Motif. Fonctionnellement, O_2Look est une extension de Motif destinée à tout ce qui est affichage à l'écran et manipulation d'objets et de valeurs O_2. Les avantages à utiliser cette interface sont bien sûr l'homogénéité pour tout affichage à l'écran, la rapidité de développement. On peut grâce à ses primitives développer rapidement une interface simple d'utilisation. La figure 5.2 nous présente certains écrans O_2Look de notre prototype dont un écran pour la saisie de la présentation d'un objet audio et un écran permettant d'écrire des requêtes OQL pour l'interrogation des présentations.

Le point faible de cette interface était la représentation des présentations. En effet, elle ne permettait aucune visualisation graphique de la présentation lors de sa construction.

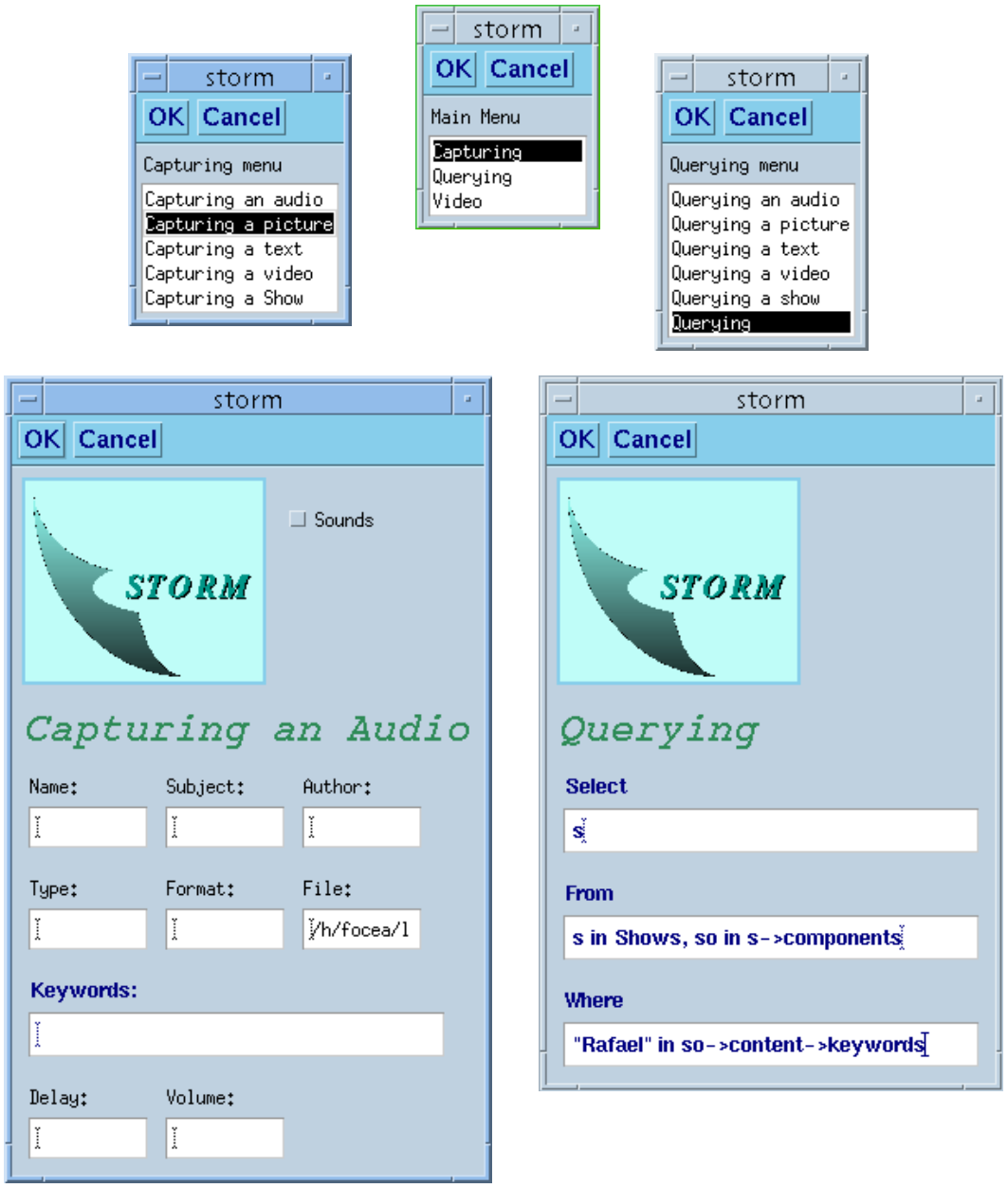

FIG. 5.2 – *Ecrans en O_2Look*

5.1.3.2 Développement à l'aide de l'outil Graphtalk

Pour pallier au manque de visualisation de la présentation de l'interface précédente, nous nous sommes intéressés à l'outil Graphtalk (une compétence de cet outil existait alors dans notre équipe) qui semblait nous offrir des possibilités intéressantes. Graphtalk est un outil pour la construction d'atelier de génie logiciel. Nous avons utilisé cette caractéristique pour la construction d'un atelier graphique personnalisable avec lequel nous pouvons modéliser nos présentations [FLM97, FLMM97]. Il s'occupe du processus de modélisation et de l'interface homme-machine de l'atelier.

Le but de notre travail a été d'établir les bases d'un environnement de haut niveau pour la construction de présentations multimédias, facile à utiliser à l'aide de graphisme. Il est capable de créer toutes sortes de présentations cohérentes par rapport au modèle STORM. Le développement de notre outil a permis finalement la formalisation des concepts du modèle à l'aide d'une modélisation objet utilisant la méthodologie OMT [Rum95] (nous présentons une partie de cette modélisation au paragraphe 5.2.3). Nous utilisons cette modélisation dans notre atelier pour créer des instances valides du modèle STORM, c'est à dire des présentations vérifiant des contraintes structurelles et temporelles définies dans le modèle. En résumé, notre environnement permet de manipuler les concepts du modèle STORM et de construire des présentations cohérentes d'un point de vue du modèle, notre atelier est décrit plus en détails en annexe A.

5.2 Extensions multimédias

Les extensions multimédias proposées sont réparties suivant différentes classes et il existe deux *bibliothèques de classes*: (1) les classes dont les instances sont appelées "objets de la base" et (2) les classes dont les instances sont les "présentations des objets de la base". Ces objets possèdent des aspects temporels sous forme d'une *Ombre Temporelle* pour être présentés. Ils sont composés et synchronisés entre eux pour des présentations plus complexes.

5.2.1 Objets multimédias

Les objets multimédias comme des images, des textes, des vidéos et des sons sont appelés les "objets de la base".

5.2.1.1 Objets de la base

Les objets multimédias sont des instances de quatres classes différentes : `Image`, `Text`, `Video` et `Audio`. Les objets des classes `Image` et `Text` sont statiques, il n'existe pas de temps spécifique associé aux objets de ces classes. Les objets des classes `Video` et `Audio` sont dynamiques par nature, ces objets représentent des données dépendantes du temps ainsi la durée est une propriété inhérente liée à ces objets. Ces classes possèdent différentes méthodes comme une méthode pour l'affichage de la donnée multimédia selon le format où elle est stockée.

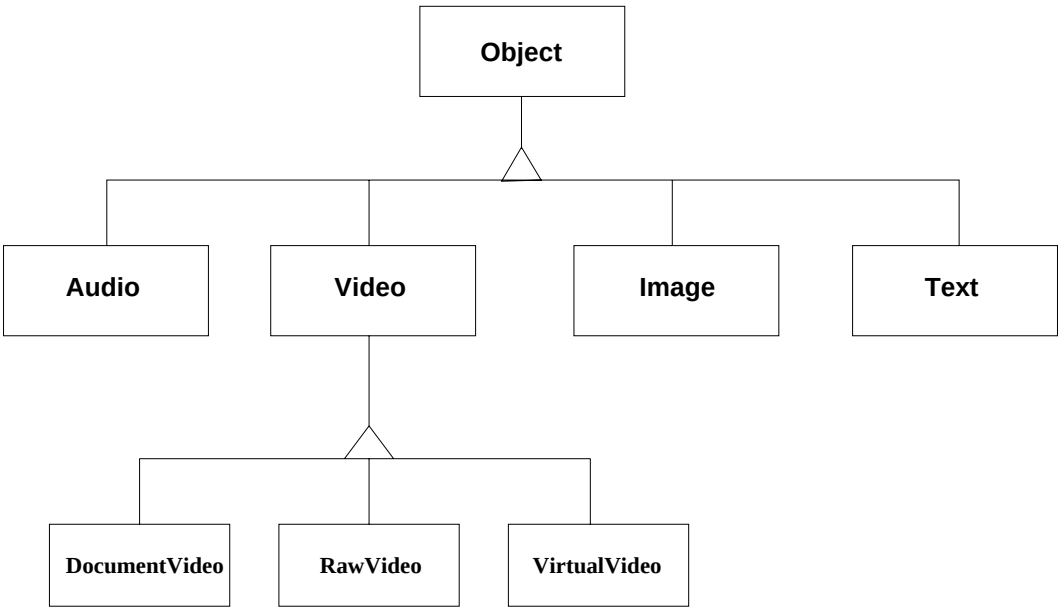

FIG. 5.3 – *Schéma de classes des objets multimédias.*

- La classe **Image** est capable de stocker différentes images selon différents formats (gif, xpm, etc.). Elle possède alors des méthodes permettant d'afficher à l'écran ces images, de modifier la taille de l'image.
- La classe **Text** stocke toutes sortes de textes et permet de les afficher suivant leur structure. Elle doit posséder des techniques d'indexation du texte sous forme de mot-clés pour pouvoir les interroger sur leur contenu et utiliser des techniques existantes de recherche d'informations.
- La classe **Audio** possède différentes méthodes pour jouer le son, changer le volume.

FIG. 5.4 – *Présentation à l'écran d'une image de Grenoble*

- La classe **Video** est une classe plus complexe. Elle se divise en sous-classes suivant le type de la vidéo. Il s'agit soit d'une *vidéo physique* appartenant à la classe `RawVideo`, soit d'une *vidéo virtuelle* de la classe `VirtualVideo` ou soit d'un *document vidéo* de la classe `DocumentVideo`.

  La classe `RawVideo` sert à stocker des vidéos physiques de différents formats. La classe `VirtualVideo` est le coeur du modèle VStorm proposé par [Loz96, Loz97, LM97]. Elle est composée d'une liste de segments vidéos de type `RawVideo`, `VirtualVideo` ou `DocumentVideo`. La vidéo virtuelle n'est pas stockée physiquement, seule la définition des différents segments est stockée. Elle est composée à l'aide d'une algèbre de vidéos définie par [WDG94, WDG95]. Les opérateurs de l'algèbre sont la *concaténation*, l'*union*, l'*intersection* et la *différence*. Enfin, la classe `DocumentVideo` spécifie la structure hiérarchique logique d'une vidéo.

5.2.1.2 Objets définis par l'utilisateur

Les objets décrits précédemment sont des objets de la base, ils permettent seulement de stocker la donnée multimédia brute. Il est difficile d'interroger des objets si ils ne possèdent pas d'attributs descriptifs. Il faut donc définir de nouveaux objets

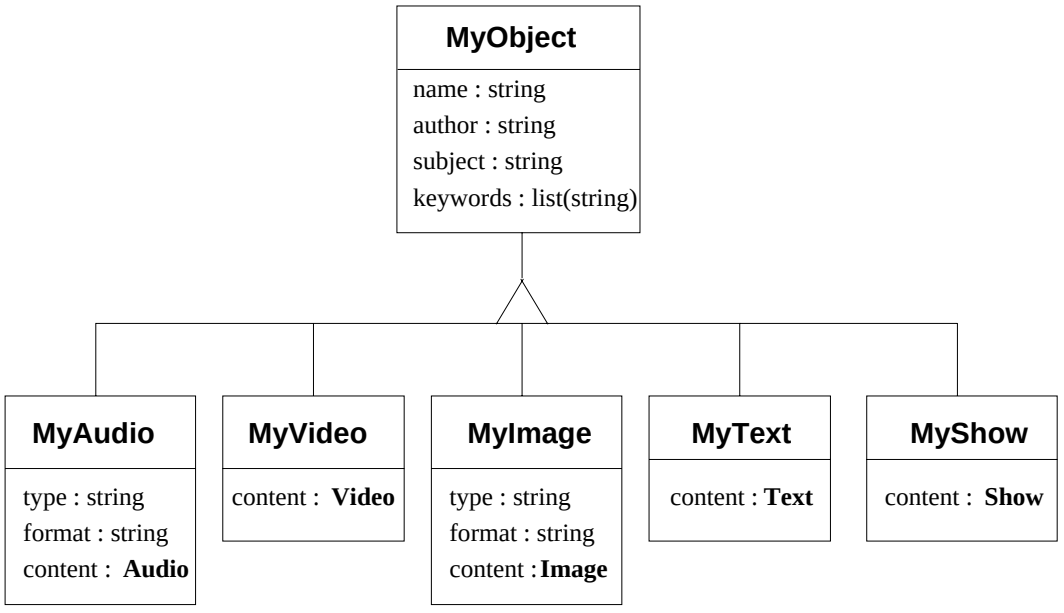

FIG. 5.5 – *Schéma de classes définies par l'utilisateur.*

faisant référence aux objets des classes `Image`, `Text`, `Video` et `Audio`, et possèdant des attributs descriptifs. On dit que ces classes sont définies par l'utilisateur parce qu'elles doivent correspondre à ses besoins.

La figure 5.5 présente un exemple de hiérarchie de classes associant à un objet multimédia différents attributs. La classe `MyObject` est la classe générique et elle contient différents attributs qui correspondent à des descriptions générales comme le *nom*, l'*auteur*, le *sujet* et une *liste de mot-clés*. Ensuite, les sous-classes `MyAudio`, `MyVideo`, `MyImage` et `MyText` possèdent leurs propres attributs correspondant à des caractéristiques spécifiques au type de média (image, texte, audio ou vidéo). Leur attribut `content` réfère à l'objet monomédia décrit.

Tous ces attributs descriptifs servent à interroger ces objets multimédias. La liste de mot-clés nous permet entre autres de décrire avec le plus de mots possibles la donnée multimédia brute. Ainsi, nous étendons notre recherche en réalisant des annotations sur ces données.

Par exemple, l'objet `Photo_Michel` de la classe `MyImage` présenté sur la figure 5.6 réfère à une image de Michel, responsable du projet STORM et décrit ses fonctions à l'aide des différents attributs.

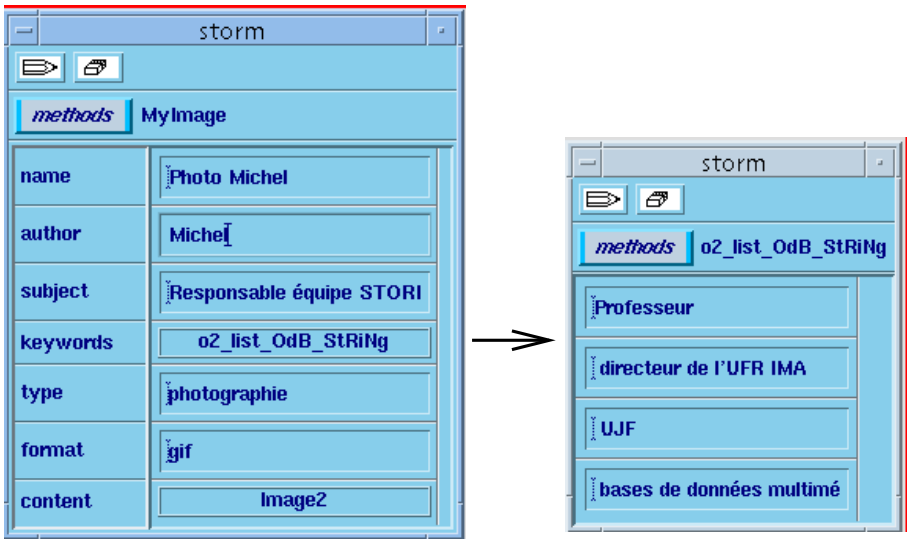

FIG. 5.6 – *Un objet de la classe MyImage*

5.2.2 Bibliothèque de classes STORM

La figure 5.7 présente la hiérarchie de classes constituant la bibliothèque de classes fournie. Elles sont prédéfinies et réutilisables (cette bibliothèque peut être utilisée dans chaque nouvelle application). Les objets de ces classes correspondent à différentes présentations multimédias. On retrouve le concept de présentations sous forme d'objets `SO` comme il est défini dans le modèle STORM présenté au chapitre 3.

La classe `SO` est la classe générique des objets STORM qui correspondent à des présentations d'objets. Toutes les classes qui héritent de la classe `SO` modélisent des présentations d'objets multimédias. Les objets de ces sous-classes possèdent des aspects temporels (*délai* et *durée*). Le délai correspond à l'attribut `delay` qui est hérité de la classe `SO`.

La racine `SO` comporte des méthodes spécifiques applicables à tout objet STORM. Une méthode `duration` permet de récupérer la valeur de la durée associée à une présentation. La méthode `start_presentation` correspond à une méthode générique de présentation d'objets. Elle analyse le contenu de l'objet (attribut `content` des objets `SO`) et construit la présentation adéquate. Elle fait appel aux méthodes `wait` et `play`. La méthode `play` est redéfinie dans chaque sous-classe de `SO` suivant le type de données, c'est elle qui présente l'objet pendant un certain intervalle de temps qui correspond à la valeur de sa durée. La méthode `wait` est définie au niveau de la classe

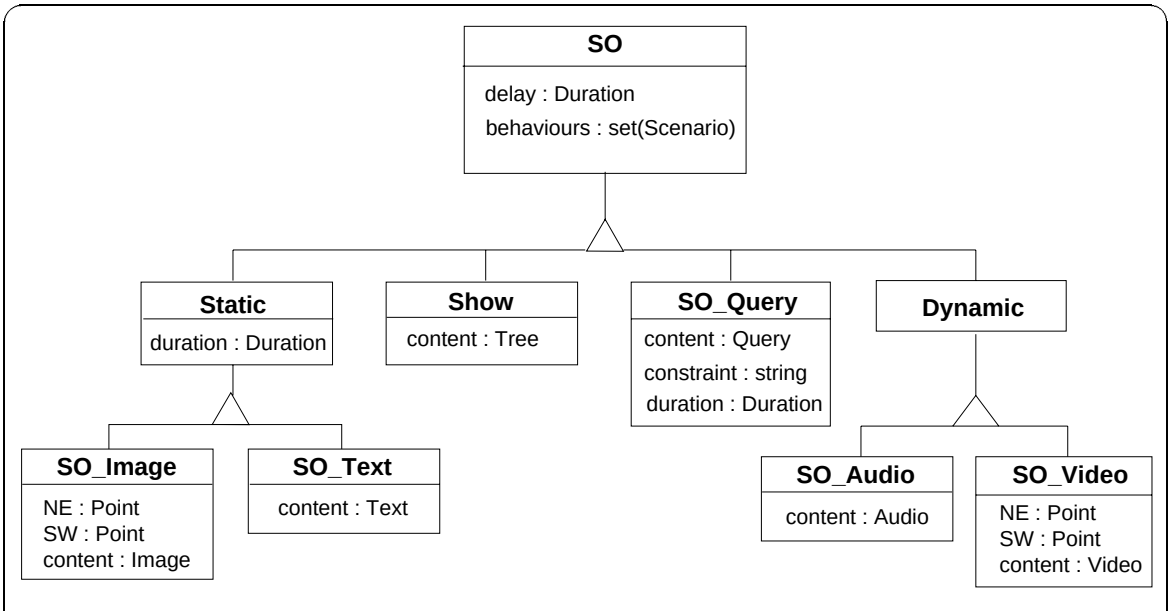

FIG. 5.7 – *Schéma de classes STORM.*

`SO`, elle est générale et applicable à tout objet STORM. Elle va lire la valeur du délai de l'objet à présenter. Si le délai est `Free`, un icone est affiché montrant que l'objet est prêt à être présenté et l'utilisateur choisit le moment où la présentation commence en cliquant sur l'icone. Si le délai est `Bound`, alors la méthode met le système en attente pendant un certain intervalle de temps qui correspond à la valeur du délai avant de démarrer la présentation.

5.2.2.1 Présentations d'objets monomédias

Les objets des classes `SO_Image`, `SO_Text`, `SO_Audio` et `SO_Video` associent à un objet monomédia de la base une ou plusieurs présentations. Ces objets STORM sont divisés en `Static` et `Dynamic`. Un objet de la classe `SO_Image` est, par exemple, la présentation d'une image. Son contenu se réfère à un objet de la classe `Image`.

Bien sur, d'autres classes STORM peuvent être créées selon les besoins de l'utilisateur. Dans ce cas, il est nécessaire de définir de nouvelles sous-classes de `SO`. Par exemple, pour pouvoir présenter un objet de la classe `MyImage`, comme nous l'avions défini au paragraphe précédent, il faut créer une classe `SO_MyImage` dont l'attribut `content` réfère à un objet de cette classe et qui héritera de la classe `Static`. De même, on peut créer les classes `SO_MyText`, `SO_MyAudio` et `SO_MyVideo`.

Plus généralement, la table 5.1 montre la correspondance entre les classes d'une

application et les présentations STORM. Une classe donnée `C` définit des objets avec attributs et méthodes spécifiques. Nous notons `Cs` l'extension de la classe `C`, appelée aussi racine de persistance. La présentation de chaque instance de `C` peut être définie à travers la classe `SO_C` qui est une sous-classe de `SO`. Chaque instance de `SO_C` est une présentation d'une instance de `C` avec des contraintes temporelles et de synchronisation. Ainsi, il est possible de construire et de stocker différentes présentations sous la racine de persistance `SO_Cs`. Une instance donnée de `C` peut être partagée par différentes présentations au travers de l'identification d'objet.

	Classes et instances	présentations STORM
Noms des classes	C	SO_C
Racines de persistance	Cs	SO_Cs

TAB. 5.1 – *Correspondance entre objets et présentations*

5.2.2.2 Présentations de plusieurs objets

Pour présenter plusieurs objets multimédias, nous les composons en spécifiant la synchronisation désirée. Ces présentations sont stockées sous forme d'instances de la classe `Show`. Son attribut `content` décrit la synchronisation entre les objets qui composent la présentation, il fait référence à un "*arbre*" dont les noeuds représentent la synchronisation et les feuilles les objets monomédias à présenter. La synchronisation peut être séquentielle ou parallèle ou une combinaison des deux. Par exemple, nous pouvons avoir une présentation séquentielle d'images, ou une présentation parallèle d'une vidéo et d'un commentaire audio, ou enfin une présentation parallèle entre une musique et une séquence de couples (image, texte) présentés en parallèle. La synchronisation peut être affinée à l'aide de différentes contraintes de synchronisation qui sont pour une séquence : {`seq_meet` et `seq_before`} et pour le parallélisme : {`par_equal`, `par_start`, `par_finish`, `par_overlap`, `par_during`} (leur sémantique respective a été définie dans le modèle STORM présenté au chapitre 3).

Un *arbre* est représenté par un objet de la classe `Tree`, il s'agit d'une classe abstraite qui regroupe les classes `Terminal_node` et `Operator_node` qui représentent respectivement les feuilles et les noeuds de l'arbre (c.f. figure 5.8). Un objet de la classe `Terminal_node` fait référence à travers son attribut `id` à un objet STORM qui fait partie de la présentation. La classe `Operator_node` est la classe abstraite pour définir les noeuds de l'arbre. Les noeuds de l'arbre représentent la synchronisation, les noeuds de type `Seq_node` et `Par_node` décrivent respectivement une synchronisation séquentielle

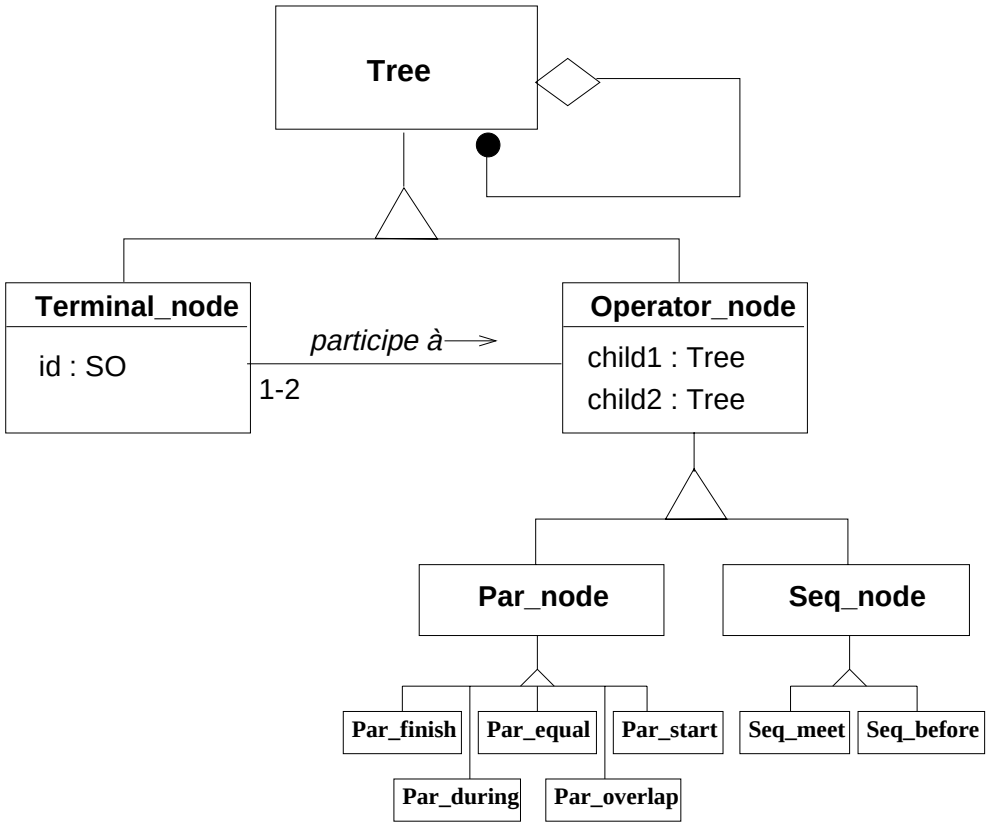

FIG. 5.8 – *La hiérarchie de classes pour définir l'arbre d'une présentation*

et parallèle sans contraintes particulières. Pour appliquer une contrainte spécifique sur la synchronisation, il suffit d'utiliser les noeuds de type `Seq_meet`, `Seq_before` ou `Par_equal`, `Par_start`, etc.

La figure 5.9 présente le déroulement dans le temps d'une présentation de Rafael, membre du projet STORM. Elle démarre par une vidéo qui dure 30 secondes, puis après un délai de 5 secondes, commencent en parallèle l'affichage d'une image de Rafael et l'écoute de son commentaire audio.

Cette présentation est stockée sous forme d'un objet `Show` présenté sur la figure 5.10, son contenu réfère à un arbre formé d'un premier noeud `Seq_before`. Il définit une synchronisation séquentielle avec une contrainte `before` entre les deux objets référencés au travers de ces deux attributs `child1` et `child2`. La contrainte `before` impose un délai entre la présentation de ces deux objets.

L'attribut `child1` fait référence à un objet de type `Terminal_node` qui contient l'objet `SO_Video_Rafael` de la classe `SO_Video`. Cet objet représente la présentation

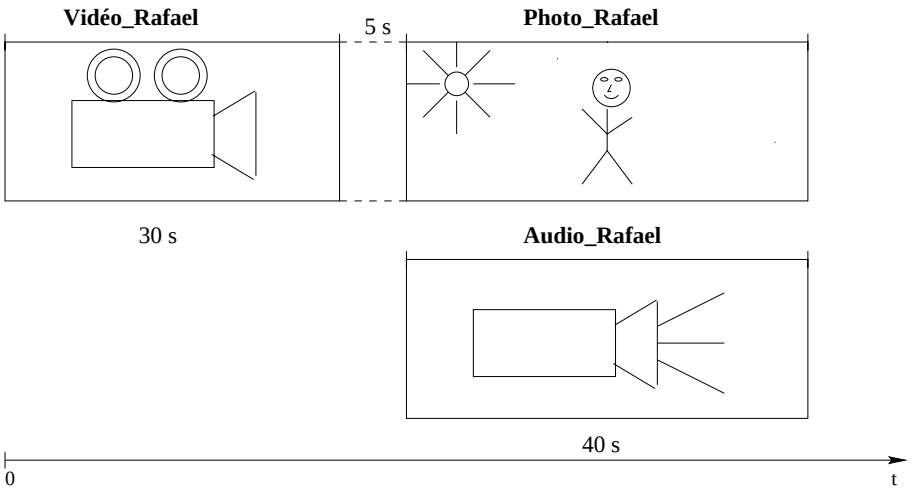

FIG. 5.9 – *Présentation de Rafael, membre du projet STORM*

de la vidéo de Rafael, avec un délai nul, elle démarre ainsi dès le début du `Show` (son délai étant aussi nul).

L'attribut `child2` fait référence à un autre noeud `Par_equal`. Il définit une synchronisation parallèle avec une contrainte `equal` entre ces attributs `child1` et `child2`. Ils font tous les deux références à un objet de type `Terminal_node`. L'objet `Terminal_node2` fait référence à la présentation de la photo de Rafael `SO_Photo_Rafael` (référencée par l'objet `Photo_Rafael` de type `MyImage`) et à la présentation d'un commentaire audio de Rafael `SO_Audio_Rafael` (référencée par l'objet `Photo_Audio` de type `MyAudio`). La contrainte `equal` impose que ces deux présentations commencent et se terminent en même temps.

5.2.2.3 Présentations basées sur des requêtes

Cette partie présente comment spécifier des présentations intentionnelles définies dans notre modèle de présentations multimédias (voir chapitre 3). Par contre, la classe `SO_Query` n'a pas encore été intégrée dans notre prototype, mais nous avons réfléchi à son implantation.

Nous reprenons un exemple de requête (`Q1`) pouvant faire partie d'une présentation intentionnelle présentant les photos de tous les membres du projet STORM sous forme d'un diaporama à raison de 3 secondes par photo. Nous donnons la définition de la requête en OQL :

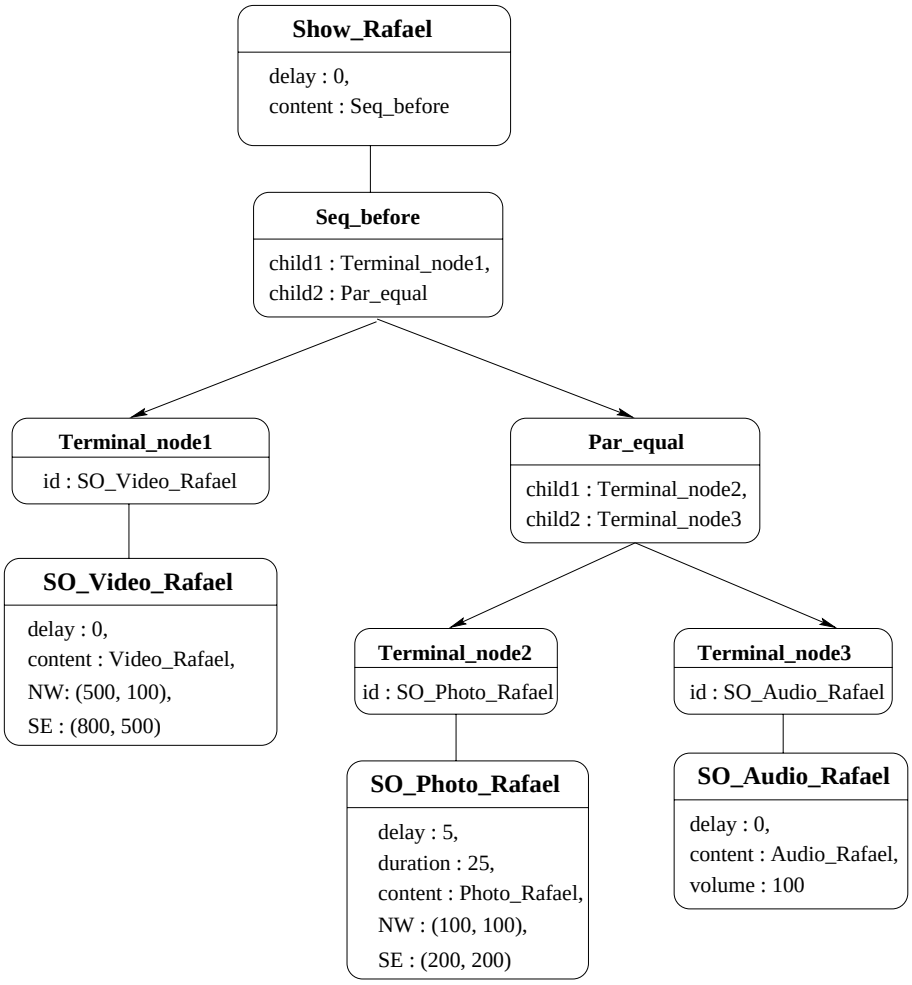

FIG. 5.10 – *Objets de la présentation de Rafael*

Q1 : Sélectionner les photos de tous les membres du projet STORM :

```
select e->photo
from   e in Les_Employés
where  e->projet->name = "STORM"
```

La requête sera exécutée au moment de jouer la présentation. Il est donc impossible de connaître a priori le nombre d'objets à présenter. Le comportement temporel de la requête est soit un comportement par défaut qui est calculé en fonction du type du résultat, soit le comportement spécifié au travers de la contrainte.

Le résultat de chacune de ces requêtes doit être un objet `SO` pour pouvoir être présenté. Ainsi la classe `SO_Query` (héritant de la classe `SO`, voir figure 5.7) permet

d'intégrer ces présentations déclaratives de façon homogène au modèle STORM.

class *SO_Query* **inherit** *SO*
tuple *(content : Query,*
constraint : string,
duration : Duration)

FIG. 5.11 – *La classe SO_Query*

L'attribut `content` de la classe `SO_Query` contient la représentation textuelle de la requête telle que définie précédemment. Les objets de la classe `SO_Query` héritent de l'attribut `delay` de la classe `SO`. L'attribut `constraint` permet d'imposer la synchronisation temporelle entre les objets résultats.

Pour l'exemple précédent `Q1`, l'objet `SO_Query` correspondant est le suivant :

objet Q1 de la classe SO_Query
tuple (content : (select e->photo
from e in Les_Employés
where e->projet->intitulé = "STORM"),
delay : 0,
constraint : "seq_meet",
duration : 3)

FIG. 5.12 – *Un objet de la classe SO_Query*

Les photos seront présentées à raison de 3 secondes chacune et la contrainte de synchronisation "seq_meet" impose qu'il n'y ait aucun délai entre leur présentation.

En ce qui concerne les types de données, l'attribut `delay` vaut toujours zéro tandis que la durée de présentation est soit inhérente au type (cas des types audio et vidéo) soit `free` (c'est à dire que la fin de la présentation est soumise à une action utilisateur).

5.2.3 Modélisation OMT des classes prédéfinies

Cette modélisation partielle à l'aide de la *méthode objet OMT* [Rum95] présente toutes les classes prédéfinies offertes par notre SGBD multimédia sous forme de bibliothèques de classes avec les différentes relations d'héritage entre elles ainsi que les

relations de composition. La figure 5.14 nous propose le modèle des objets nommé "*Présentation*" (voir le détail des notations OMT en annexe B) .

Le modèle des objets *Présentation* peut être divisé en *Sous-Systèmes* selon la figure 5.13. Le sous-système *Structure Algébrique* contient les classes qui définissent la structuration des données multimédias; le sous-système *STORM* contient les classes qui composent la bibliothèque de classes STORM du modèle; le sous-système *Vidéo* contient les classes qui font l'implantation d'une vidéo dans le système; le sous-système *Structure Vidéo* contient les classes qui implantent la structuration des données vidéo dans le système.

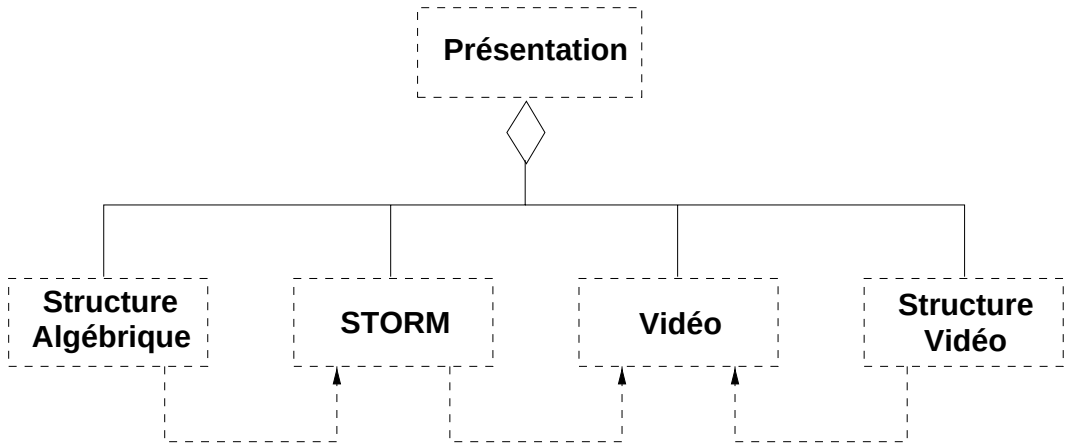

FIG. 5.13 – *Diagramme de Sous-Systèmes*

Nous présentons aussi la composition de la donnée vidéo comme nous l'avions défini précédemment. Elle peut être de trois types : `DocumentVideo`, `RawVideo` et `VirtualVideo`. La vidéo virtuelle est composée de `videoInterval` qui est soit un `RawVideoInterval`, soit un `VirtualVideoInterval`. Un objet `DocumentVideo` définit la structure hiérarchique de la vidéo, il est composé d'un ensemble de `Sequence`, composées elles-mêmes d'un ensemble de `Scene` et elles-mêmes d'un ensemble de `Shot`.

5.3 Gestion des présentations multimédias

Nous avons vu au cours du paragaphe précédent comment nous stockions nos présentations multimédias sous forme de différents objets STORM. Nous allons voir maintenant comment nous les manipulons en décrivant : (1) comment construire une présentation en différentes étapes; (2) comment permettre de jouer cette présentation; et

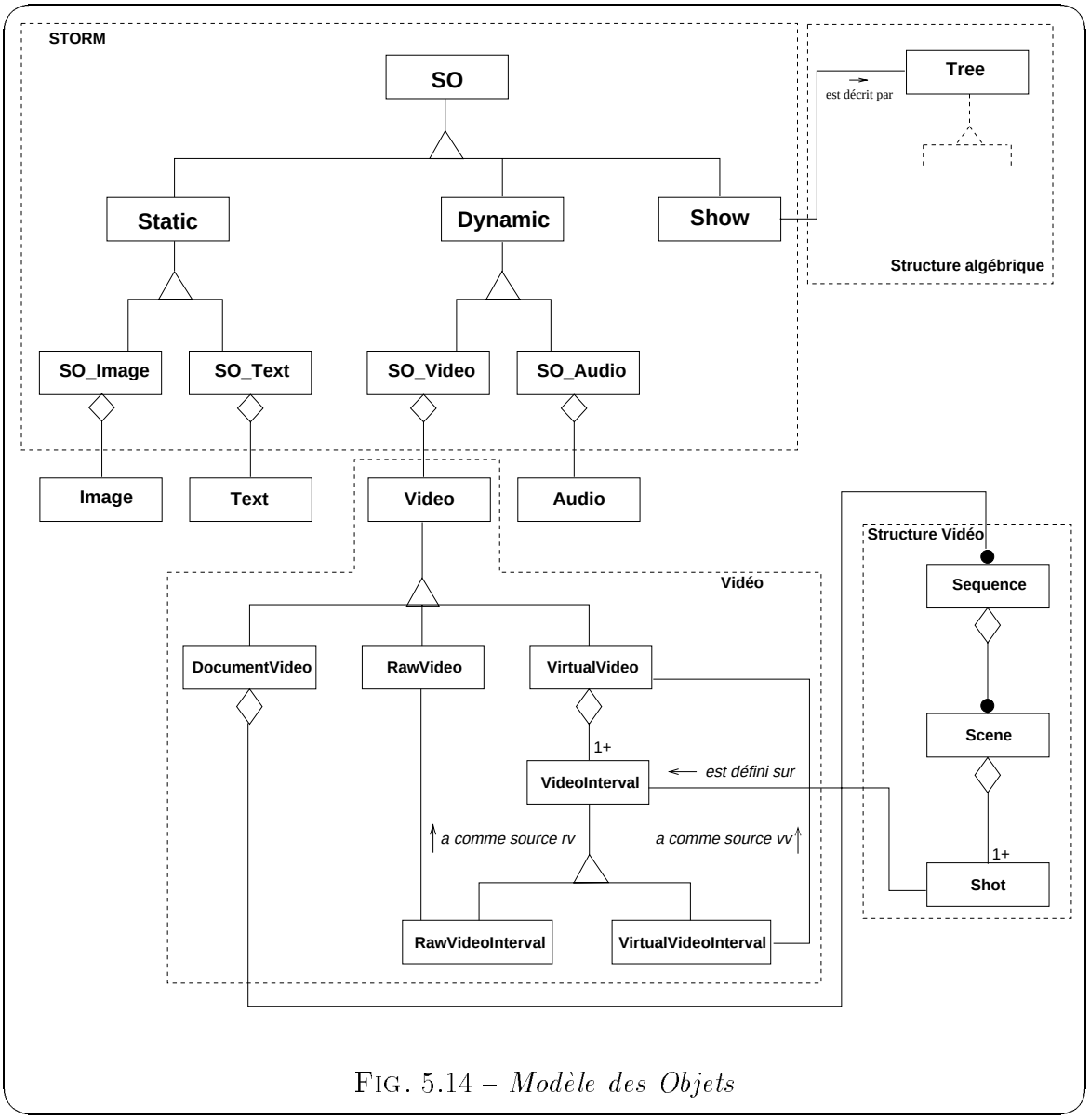

FIG. 5.14 – *Modèle des Objets*

enfin (3) comment interroger nos présentations pour les rejouer par exemple. Toutes ces fonctionnalités sont offertes par notre SGBD multimédia. Pour montrer leur faisabilité, nous avons étendu le SGBD O_2 pour qu'il offre ces différentes fonctionnalités multimédias.

5.3.1 Construire une présentation

Dans le modèle STORM, nous avions présenté comment créer une présentation STORM d'un point de vue de l'utilisateur. Dans ce paragraphe, nous nous intéres-

sons à la construction d'une présentation **du point de vue du gestionnaire de présentations**. Elle se fait en trois étapes et s'effectue suite aux différents choix de l'utilisateur lors de la création de la présentation (décrite au paragraphe 3.7) :

1. Créer ou rechercher les objets de la base qui feront partie de la présentation,
2. Créer les objets STORM correspondant (de la classe `SO`) avec des aspects temporels et spatiaux,
3. Etablir une synchronisation parallèle ou séquentielle entre ces objets, et créer l'objet STORM qui correspond à la présentation voulue.

Ensuite, une fois construite, la présentation peut être jouée et devenir persistante. Nous allons décrire ces trois étapes de construction sur un exemple. Reprenons l'exemple de la présentation multimédia de la ville de Grenoble dont nous avions décrit la création au paragraphe 3.7. Il s'agit d'une présentation parallèle d'une musique, d'une séquence (Image, Vidéo) et d'un texte. Nous avons besoin de quatre objets monomédias pour construire cette présentation. Le temps de la musique correspond au temps de la présentation. La durée du texte correspond à la durée de la séquence de l'image et de la vidéo. La construction de cette présentation par le gestionnaire de présentations se fait selon les étapes suivantes :

- Premièrement, le gestionnaire récupère les objets de la base qui sont choisis par l'utilisateur et qui seront présentés au moment de jouer la présentation. L'image choisie a pour nom "Photo_Grenoble", elle appartient à un objet de la classe `MyImage`. La musique, la vidéo et le texte appartiennent respectivement à trois objets des classes `MyAudio`, `MyVideo` et `MyText`.
- Pour chaque objet sélectionné, le gestionnaire crée les objets STORM correspondant dont le contenu fait référence à l'identificateur de l'objet à présenter. Ainsi les objets de la base ne sont pas dupliqués, et ils peuvent être référencés par différents objets STORM correspondant à différentes présentations. Chaque objet STORM possède une *Ombre Temporelle* (un délai et une durée) et des coordonnées dans l'espace pour les données affichables à l'écran. L'utilisateur a saisi ces données suivant le déroulement dans le temps et sur l'écran qu'il souhaite. Dans ce cas-là, on va créer quatre objets de type `SO_MyAudio`, `SO_MyImage`, `SO_MyText` et `SO_MyVideo`.
- Enfin, le gestionnaire va créer la synchronisation entre les objets STORM, ainsi il pourra jouer la présentation en prenant en compte les aspects temporels de

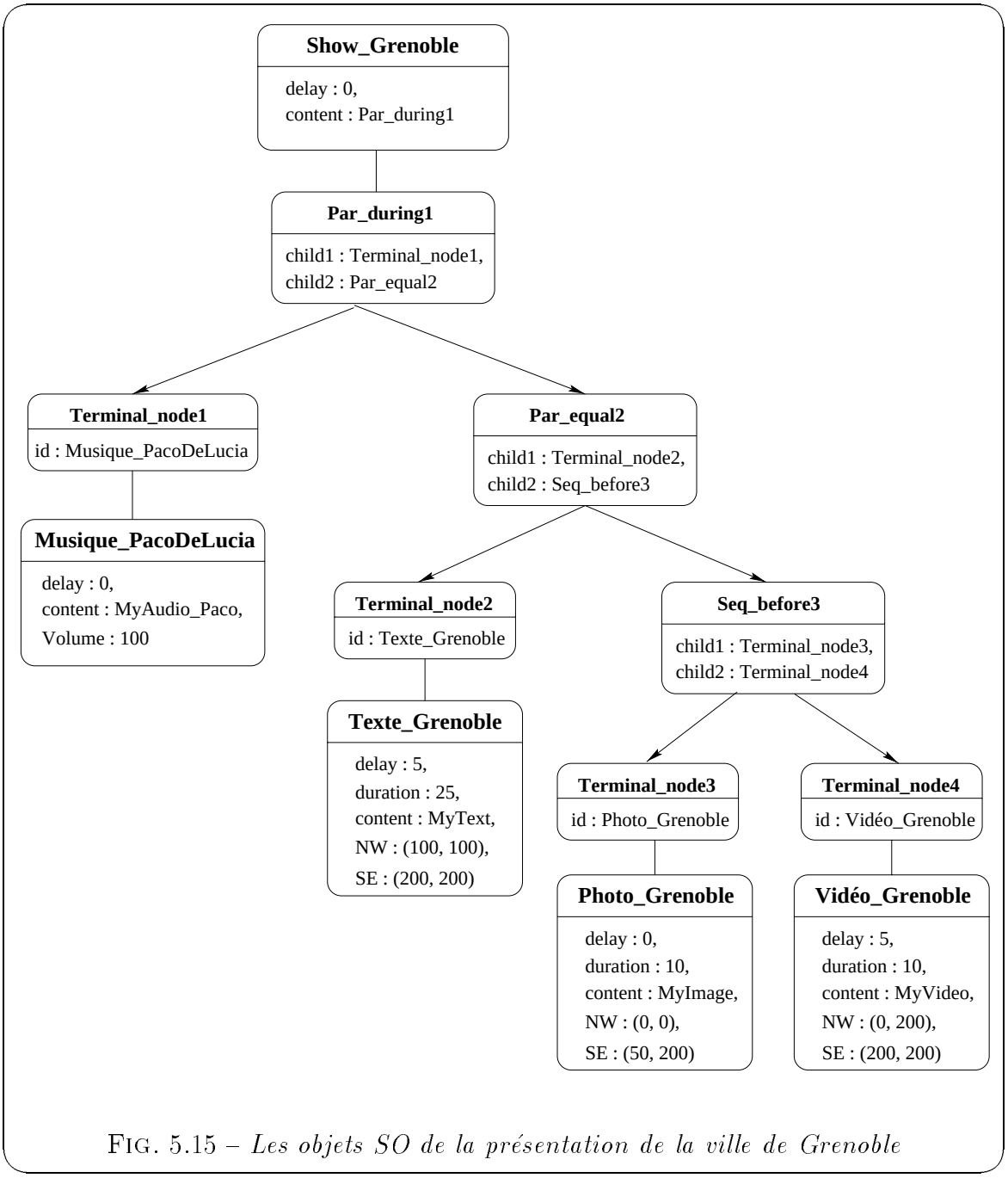

FIG. 5.15 – *Les objets SO de la présentation de la ville de Grenoble*

chaque objet et le type de synchronisation entre eux. La synchronisation peut se décrire par l'expression suivante : *par_during(Musique_PacoDeLucia, par_equal(Texte_Grenoble, seq_before(Photo_Grenoble, Vidéo_Grenoble)))* (la sémantique de ces contraintes a déjà été décrite au paragraphe 3.7).

Le gestionnaire va créer l'arbre qui définit la synchronisation entre les objets `SO` qui composent la présentation. Cet arbre est référencé par le contenu d'un

objet de la classe `Show` qui est créé pour représenter la présentation. Ainsi, elle pourra être interrogée et jouée. La figure 5.15 présente cet objet `Show_Grenoble`, ainsi que les différents objets qui composent la présentation et les différents objets qui forment l'arbre. Le noeud principal de cet arbre est `Par_during1`, ses fils sont `Terminal_node1` et `Par_equal2`. La synchronisation entre ces deux fils sera parallèle avec une contrainte de synchronisation de type `during`. Ainsi de suite, les différents objets définissent la synchronisation entre les objets `SO` `Musique_PacoDeLucia`, `Texte_Grenoble`, `Photo_Grenoble` et `Vidéo_Grenoble`.

5.3.2 Jouer une présentation

La construction de la présentation "`Show_Grenoble`" a permis de stocker cette présentation sous forme d'un objet STORM dans la base. Le gestionnaire permet aussi de jouer cette présentation. Il suffit d'appliquer la méthode "`start_presentation`" de la classe `SO` sur cet objet. Il s'agit d'une méthode prédéfinie de la classe `SO` et toutes ses sous-classes en héritent. Elle permet de jouer chaque présentation en prenant en compte les aspects temporels des objets qui la composent et la synchronisation définie entre eux. Cette méthode s'applique elle-même sur tous les objets STORM qui composent le show en parallèle ou en séquence suivant la synchronisation.

5.3.3 Interroger une présentation

Dans cette partie, nous montrons comment nous pouvons interroger nos présentations multimédias selon différents critères. L'interrogation est un des points crucial pour le développement des SGBD multimédias et un des aspects en plus par rapport aux autres systèmes. En effet, les langages de requêtes permettent de rechercher des données multimédias, mais aussi des présentations selon différents critères : soit les *aspects descriptifs* des objets (attributs, mot-clés), soit les *aspects temporels* (délai et durée) ou encore les *aspects de synchronisation* entre les objets de la présentation. A l'heure actuelle, aucun système auteur n'est capable de retrouver les présentations multimédias existantes selon ces critères. Ceci conforte notre approche qui permet de stocker nos présentations sous forme d'objets dans la base et permettre à l'utilisateur de les retrouver facilement pour pouvoir les jouer ou les modifier ou encore les réutiliser.

Nous utilisons le langage OQL pour garder ses avantages et nous proposons des extensions sous forme de méthodes pour certains types d'interrogation. Dans les deux

premiers cas, nous interrogeons nos présentations sur des aspects descriptifs, il peut s'agir d'attributs, d'une liste de mot-clés ou de ses aspects temporels (délai et durée). Dans l'autre cas, nous nous intéressons à la synchronisation entre eux.

Les racines de persistance utilisées dans nos différentes requêtes sont les suivantes :

```
SO_MyAudios : set(SO_MyAudio),
MyAudios    : set(MyAudio),
SO_MyImages : set(SO_MyImage),
SO_MyVideos : set(SO_MyVideo),
MyImages    : set(MyImage),
MyShows     : set(MyShow),
Shows       : set(Show)
```

5.3.3.1 Interrogation sur des aspects descriptifs

Certains objets de la base possèdent différents attributs qui permettent de les décrire : `name`, `subject`, `keywords`, etc. Ils offrent ainsi une aide à l'interrogation de ces objets.

Dans la requête `Q1`, l'utilisateur utilise simplement le nom de la présentation pour la rechercher.

Q1 : rechercher le show dont le nom est "Membres du projet STORM" :

Select *s*
from *m in MyShows*
where *m->name = "Membres du projet STORM"* **and**
m->content = s

Une liste de mot-clés (`keywords`) peut être associée à un objet pour les décrire; il s'agit d'une liste exhaustive. Elle permet de donner une description plus précise de la présentation par rapport à ces attributs. La requête `Q2` permet donc de retrouver toutes les présentations qui parlent de "Rafael".

Q2 : rechercher toutes les présentations qui parlent de "Rafael" :

Select *s*
from *s in Shows, so in s->components*
where *"Rafael" in so->content->keywords*

La méthode "`list(SO) components()`" de la classe `Show` permet d'extraire les objets STORM qui composent la présentation et retourne une liste composée de ces objets.

Nous pouvons aussi interroger les objets STORM sur leurs aspects temporels `delay` et `duration`.

Q3 : rechercher toutes les photos de Michel qui sont présentées pendant moins d'une minute :

Select *i->name*
from *i in MyImages, si in SO_MyImages*
where *i->content = si* **and**
si->duration < 60 **and**
i->title = "Photo_Michel"

Dans ce cas-là, on utilise tout simplement la valeur de l'attribut `duration` de l'objet `si` de la classe `SO_MyImage`.

Q4 : rechercher les titres des musiques qui durent moins de 10 minutes (ou 600 secondes) et qui sont composées par "Beethoven" :

Select *a->name*
from *a in MyAudios, sa in SO_MyAudios*
where *sa->content = a* **and**
sa->duration < 600 **and**
a->author = "Beethoven"

Cette requête utilise la méthode `duration()` de la classe `SO_Myaudio`. En effet, l'objet `sa` est un objet dynamique (il s'agit d'un son), il possède une durée inhérente à sa nature. Cette méthode renvoie la durée du temps d'écoute de la musique.

Q5 : rechercher tous les shows qui présentent une vidéo d'Agnès (membre du projet STORM) pendant plus de 3 minutes :

Select *s*
from *s in Shows, sv in s->components, sv in SO_MyVideos*
where *sv->duration > 180* **and**
sv->content->subject = "Vidéo d'Agnès"

Q6 : rechercher tous les shows qui durent plus de 15 minutes et qui parlent de "Bases de données multimédias" :

Select *s*
from *s in Shows, m in MyShows*
where *s->duration > 900* **and**
"Bases de données multimédias" in m->keywords **and**
m->content = s

Les requêtes `Q5` et `Q6` utilisent la méthode `duration()` de la classe `Show`. Cette méthode calcule la durée totale du show suivant les valeurs et les durées de ses composants et la synchronisation entre eux.

5.3.3.2 Interrogation sur la synchronisation entre les objets

Dans cette partie, nous allons étudier comment étendre OQL pour pouvoir interroger une présentation sur la synchronisation entre les objets qui la composent. La synchronisation est de type séquentiel ou parallèle.

Pour adapter le langage OQL à ce type de requêtes, nous définissons une méthode `synchro` au niveau de la classe `Show` qui renvoie le type de synchronisation, comme `par` ou `seq` entre deux objets de cette présentation. Le type de synchronisation peut être aussi : {`seq_meet`, `seq_before`} et {`par_equal`, `par_start`, `par_finish`, `par_overlap`, `par_during`}.

Les deux exemples de requêtes qui suivent utilisent cette méthode pour interroger les présentations sur la synchronisation entre leurs objets composants. La première requête recherche toutes les présentations d'une image (objet `SO_MyImage`) de Michel faisant partie d'une présentation multimédia présentant cette image suivie immédiatement par une image de Hervé. On recherche une synchronisation séquentielle entre deux images avec une contrainte `seq_meet` pour qu'elles se suivent immédiatement (donc sans délai entre leurs présentations).

Q7 : rechercher toutes les photos de Michel qui sont suivies immédiatement par une photo de Hervé :

Select *pg*
from *s in Shows, pg in s->components, pg in SO_MyImages,*
pp in s->components, pp in SO_MyImages

where *pg.subject = "Photo de Michel"* **and**
pp.subject = "Photo de Hervé" **and**
s->synchro(pg, pp) = "seq_meet"

La deuxième requête recherche des couples (Vidéo, Audio) présentant la ville de Grenoble. La vidéo et le son sont présentés en parallèle avec une contrainte `par_equal`.

Q8 : trouver tous les couples (Vidéo, Audio) décrivant la ville de Grenoble dans des présentations :

Select *tuple(Video: vg, Audio: ag)*
from *s in Shows, vg in s->components(), vg in SO_MyVideos,*
ag in s->components(), ag in SO_MyAudios
where *vg.subject = "Grenoble"* **and**
ag.subject = "Grenoble" **and**
s->synchro(vg, ag) = "equal"

Bilan - L'interrogation des présentations à l'aide du langage OQL se fait facilement avec l'aide de différentes méthodes rattachées aux objets `SO`. Il faudrait coupler cette interrogation avec des systèmes de recherche basés sur le contenu pour bénéficier de la sémantique de ces données [AC95, FSNA95, ABF+95]. Dans le cadre de notre projet, [Loz97, LM97] propose une indexation du contenu de la donnée vidéo pour permettre d'interroger des vidéos physiques stockées dans la base, mais aussi virtuelles.

5.4 Gestion des comportements d'une présentation

Dans le chapitre 4, nous avons défini notre modèle de comportements d'une présentation multimédia. Il propose d'associer à une présentation une *Ombre Comportementale* qui permet de définir différents comportements pour une présentation d'un objet. On appelle *comportement d'une présentation d'un objet*, l'ensemble des actions qu'il est susceptible d'entreprendre suite à des événements lors de sa présentation. Nous choisissons de réaliser l'*Ombre Comportementale* sous forme d'un ensemble d'*objets scénarios*. Ces objets sont des instances des classes `Scenario` appartenant à la **bibliothèque de classes de comportements** que nous proposons à l'utilisateur.

Dans une deuxième partie, nous décrirons comment nous utilisons la technologie des SGBD actifs pour exécuter ces différents scénarios.

5.4.1 Bibliothèque de classes de comportements

Comme nous venons de décrire la bibliothèque de classes réalisant les extensions multimédias, nous implantons les classes scénarios à l'aide d'une **bibliothèque de classes de comportements** (présentée dur la figure 5.16). Les instances des classes scénarios représentent les comportements liés à une présentation. Ces classes correspondent aux différents comportements que nous avons défini au chapitre 4, mais l'utilisateur peut en créer d'autres suivant les besoins des applications.

La classe `Scenario` est la classe générique. Les scénarios sont décomposés en scénarios externes et internes suivant s'ils sont déclenchés par un événement externe ou interne, ainsi la classe `Scenario` possède deux sous-classes `External_Scenario` et `Internal_Scenario` comme le montre la figure 5.16.

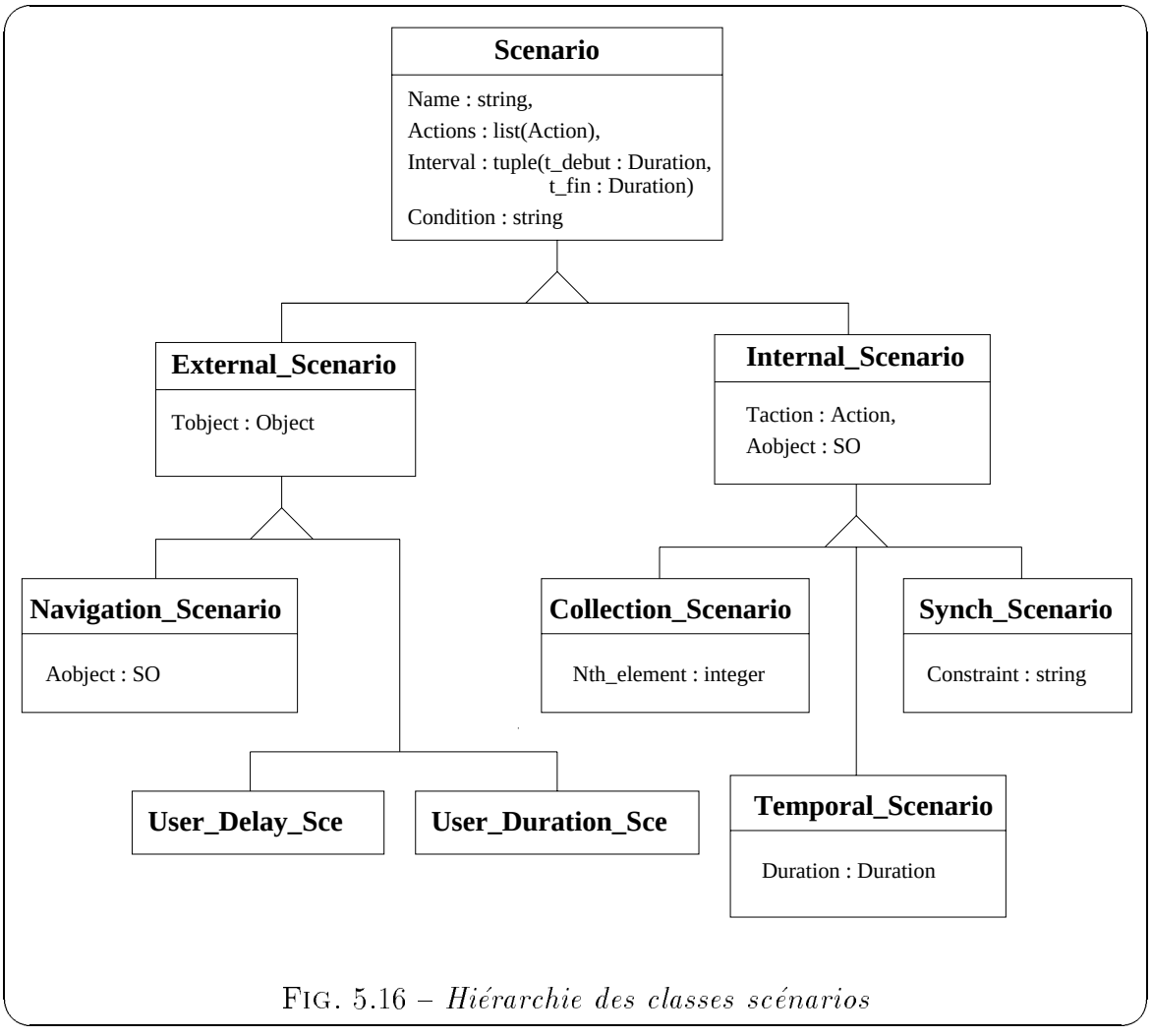

FIG. 5.16 – *Hiérarchie des classes scénarios*

Ces objets scénarios décrivent le comportement d'un objet `SO`, ils sont rattachés

à cet objet par l'attribut `behaviours`. Cet attribut réfère à un ensemble d'objets scénarios :

class *SO* {
public:
d_Ref<Duration> delay,
d_Set<d_Ref<Scenario> > behaviours}

La classe `SO` possède deux méthodes pour gérer les comportements `start_behaviour` et `end_behaviour`, la première permet d'activer les différents comportements liés à l'objet au début de sa présentation. Par exemple, on pré-charge les objets qui sont liés à ses comportements. En fait, elle va activer un ensemble de règles actives qui sont capables de réagir aux événements et d'exécuter le scénario, c'est à dire un ensemble d'actions à réaliser (voir paragraphe 5.4.4 pour l'exécution des scénarios à l'aide de règles actives). Au contraire, la méthode `end_behaviour` va desactiver ces règles actives à la fin de la présentation de l'objet, les comportements liés à cet objet n'étant plus valides.

La classe `Scenario` suivante possède différentes sous-classes qui correspondent à différents types de comportements. Chaque objet de ces classes définit un scénario possible de la présentation.

- **La classe** *Scenario*

class *Scenario* {
public:
d_String Name,
d_List<d_Ref<Action> > Actions,
d_Interval Interval,
d_String Condition }

Un objet scénario est défini par un *identificateur* et appartient à la classe générique `Scenario` possèdant un attribut `name`, un attribut `Actions` qui décrit la liste des actions déclenchées par l'événement. Une action correspond à l'exécution d'une méthode d'un objet. L'attribut `Interval` correspond à l'intervalle de validité du comportement. L'attribut `Condition` définit la condition nécessaire

à l'exécution de l'ensemble des actions, elle correspond en fait à tous les événements (correspondant à des appels de méthodes) qui ont dû se produire avant l'événement déclencheur du comportement.

La classe `Scenario` possède la méthode `rules()` qui va générer à partir de l'objet scénario un ensemble de règles actives. Les identifiants de ces règles sont stockés au niveau de l'objet (sous forme d'un attribut privé), ainsi la méthode `start_behaviour()` de la classe `SO` les utilisera pour activer les règles actives.

La classe `Action` permet de définir les actions liées à l'événement, sa structure est la suivante :

class *Action* {
    public:
        d_Ref<SO> Aobject,
        int action_number }

La classe `Action` possède une méthode `link()` qui fait le lien entre le numéro de l'action (`action_number`) et la méthode qui sera exécutée sur l'objet `Aobject` à l'aide d'une table de correspondances.

- **La classe** *External_Scenario* **et ses sous-classes** *Navigation_Scenario*, *User_Delay_Sce* **et** *User_Duration_Sce*

class *External_Scenario* **: public** *Scenario* {
    public:
        d_Ref<Object> Tobject}

La classe `External_Scenario` est la classe générique pour les scénarios déclenchés par des événements externes, ses deux sous-classes `Navigation_Scenario` et `User_Scenario` décrivent deux types particuliers d'interaction avec l'utilisateur. L'attribut `Tobject` de cette classe correspond à l'objet déclencheur de l'événement. Comme il s'agit d'un événement utilisateur, il est associé à un objet ou un bouton sur lequel l'utilisateur pourra cliquer.

class *Navigation_Scenario* **: public** *External_Scenario* {
    public:
        d_Ref<SO> Aobject }

L'attribut `Aobject` correspond à l'objet affecté au cours du scénario. Cet objet sera joué lorsque l'événement survient. Ces scénarios de navigation permettent d'établir des liens navigationnels entre un objet d'une présentation et une autre présentation.

Les deux sous-classes `User_Delay_Sce` et `User_Duration_Sce` ont pour instances des objets décrivant respectivement un *délai interactif* et une *durée interactive*. La valeur de l'intervalle de validité du *délai interactif* (ou de la *durée interactive*) correspond à la valeur de l'attribut `Interval`.

- **La classe** *Internal_Scenario* **et ses sous-classes** *Collection_Scenario*, *Synch_Scenario* **et** *Temporal_Scenario*

class *Internal_Scenario* : **public** *Scenario* {
public:
d_Ref<Action> Taction,
d_Ref<SO> Aobject }

La classe `Internal_Scenario` est la classe générique pour les scénarios déclenchés par des événements internes, elle possède trois sous-classes : `Collection_Scenario`, `synch_Scenario` et `Temporal_Scenario`. L'attribut `Taction` correspond à l'action déclenchante du scénario.

class *Collection_Scenario* : **public** *Internal_Scenario* {
public:
int Nth_element }

L'attribut `Nth_element` définit le numéro de l'élément dans le résultat de la requête (il s'agit d'une liste) qui, associé à l'action déclenchante, produira le scénario (il s'agit de l'application de la méthode définie par l'action sur le n-ième objet de la requête).

class *Synch_Scenario* : **public** *Internal_Scenario* {
public:
d_String Constraint }

L'attribut `Constraint` définit la contrainte de synchronisation entre l'objet `SO` qui possède cet objet scénario et l'objet référencé par l'attribut `Aobject`.

class *Temporal_Scenario* : **public** *Internal_Scenario* {
public:
d_Ref<Duration> Duration }

L'attribut `duration` définit le temps de déclenchement du scénario après l'action déclenchante.

5.4.2 Exemple d'utilisation de ces classes

Ces différentes classes vont nous permettre d'instancier les différents comportements liés à une présentation. Montrons sur différents exemples, les objets scénarios à créer pour définir un comportement particulier.

5.4.2.1 Définition d'un délai interactif

L'objet `Delay_Photo_Hervé` de la classe `User_Delay_Sce` (présenté sur la figure 5.17) permet d'associer à un objet `SO`, dont le délai a une valeur `bound` de 40s, un délai interactif dont l'intervalle de validité est [10s, 30s].

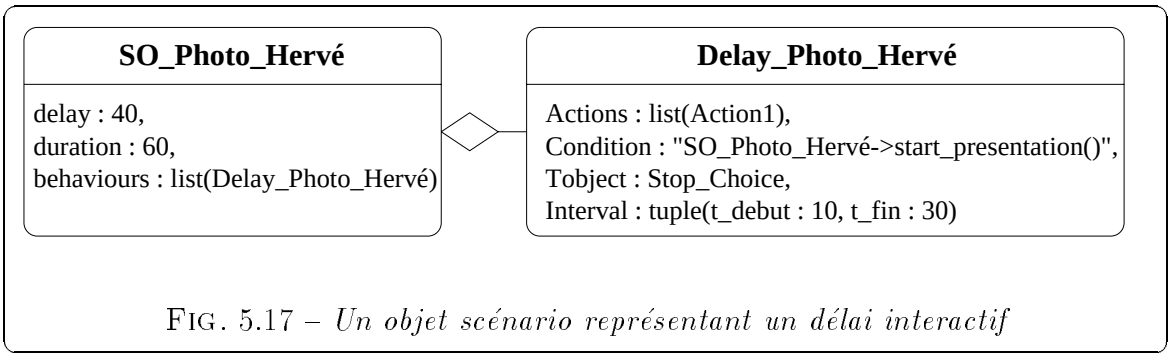

FIG. 5.17 – *Un objet scénario représentant un délai interactif*

L'action `Action1` démarre l'observation de l'objet SO lié à ce comportement (on lui applique la méthode `play()`). Cette action est exécutée si l'utilisateur a cliqué sur l'objet `Stop_Choice` et si la condition est satisfaite, c'est à dire que l'événement "SO_Photo_Hervé->start_presentation()" s'est produit (la présentation de la photo d'Hervé a bien démarré).

5.4.2.2 Définition d'un lien navigationnel

Reprenons l'exemple de la figure 4.12 qui présente le déroulement de la présentation des membres du projet STORM avec un lien navigationnel au niveau de l'image

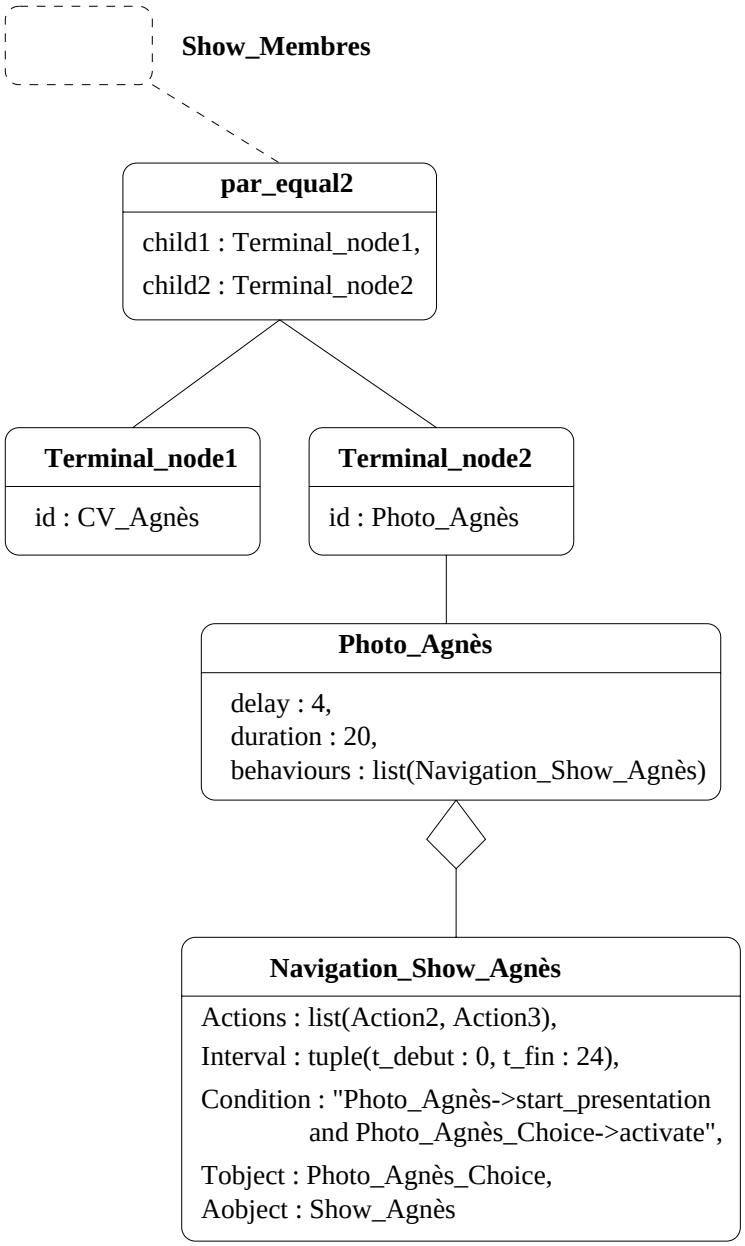

FIG. 5.18 – *Un objet scénario représentant un lien navigationnel*

`Photo_Agnès`. Si l'utilisateur clique sur cette photo (en fait il clique sur l'objet `Photo_Agnès_Choice` correspondant à un bouton transparent qui est activé au début de la présentation à l'aide de la méthode `activate`), il déclenche une autre présentation présentant Agnès (membre du groupe).

Ce lien est représenté à l'aide de l'objet `Navigation_Photo_Agnès` de la classe

`Navigation_Scenario` présenté sur la figure 5.18. Cet objet est référencé par l'objet `Photo_Agnès` de type `SO` qui fait partie de la présentation plus complexe `Show_Membres`.

L'action `Action2` stoppe la présentation `Show_Membres` et l'action `Action3` démarre la présentation `Show_Agnès` si la condition est satisfaite, c'est à dire que les deux événements "Photo_Agnès->start_presentation" (démarrage de la présentation de la photo d'Agnès) et "Photo_Agnès_Choice->activate" (activation du bouton par l'utilisateur) ont eu lieu.

5.4.2.3 Contrainte de synchronisation spécifique

Reprenons l'exemple de la figure 4.13 où l'on cherche à exprimer une contrainte de synchronisation spécifique `par_finish` entre les objets `Texte_PrésentationProjet` et `Image_chef`. Ce comportement est spécifique à ces deux objets, chacun possède un comportement décrivant cette contrainte de synchronisation, comme le montre la figure 5.19.

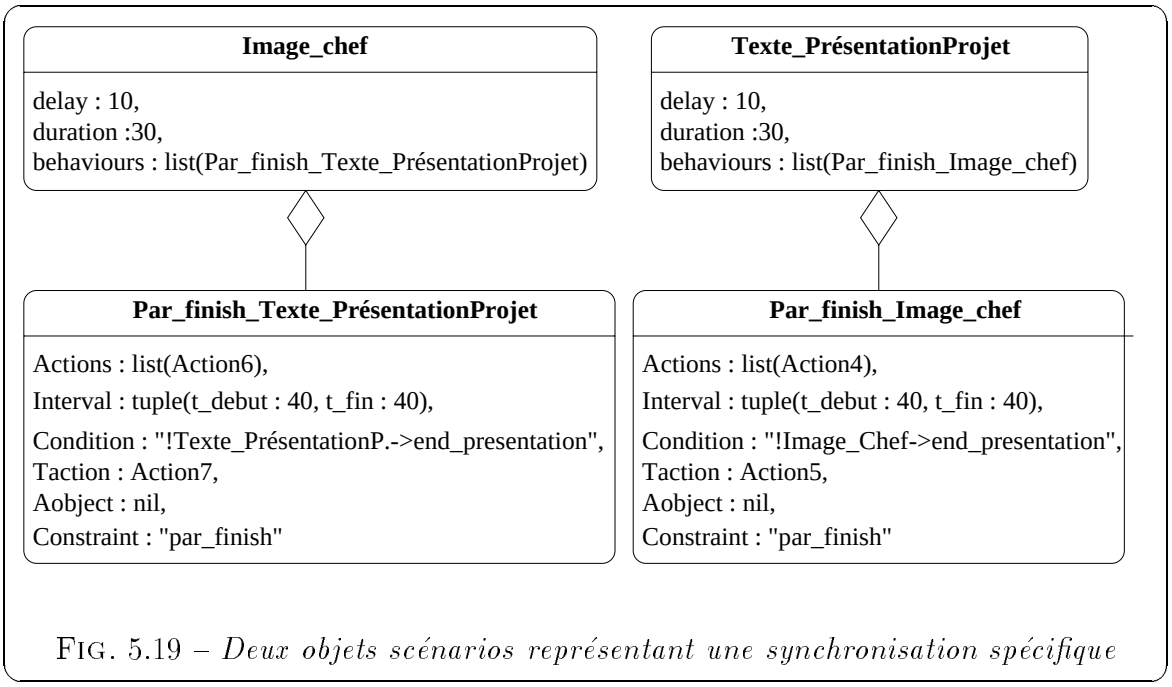

FIG. 5.19 – *Deux objets scénarios représentant une synchronisation spécifique*

Les deux objets scénarios `Par_finish_Image_Chef` et `Par_finish_Texte_PrésentationProjet` représentent ces comportements et sont des objets de la classe `Synch_Scenario`.

L'action `Action4` déclenche la fin de la présentation de l'image avec la méthode `end_presentation()` sur l'objet `Photo_Agnès`. L'action `Action5` est l'action déclen-

chante de l'événement, il s'agit de la fin de la présentation du texte. Ces actions sont exécutées seulement si la condition est satisfaite, c'est à dire que l'événement "Image_Chef->end_presentation" ne s'est pas produit (la notation "!" représente la négation sur l'événement), en d'autres termes la fin de la présentation de l'image du chef n'a pas eu lieu. Ces actions réalisent la vérification de la contrainte de synchronisation `par_finish`.

Les actions `Action6` et `Action7` sont inversées par rapport au premier comportement.

5.4.2.4 Les comportements d'une requête d'une présentation

Reprenons ici l'exemple de la présentation basée sur une requête `SO_Query_Membres` décrit par la figure 4.14. Cette présentation permet de présenter, à raison de 10 secondes, les photos et Curriculum Vitae des membres du projet STORM.

- *Evénement temporel*

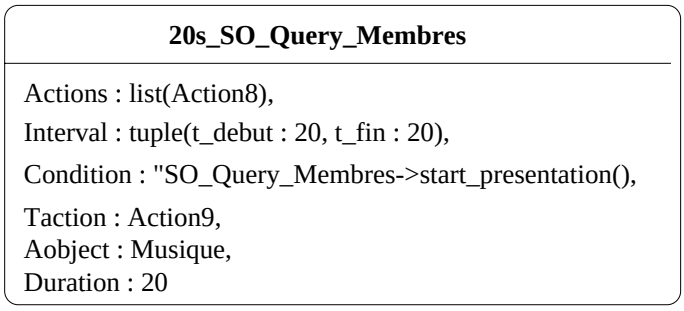

FIG. 5.20 – *Un objet scénario représentant une action suite à un événement temporel*

L'objet `20s_SO_Query_Membres` de la classe `Temporal_Scenario`, présenté sur la figure 5.20, définit qu'au bout de 20 secondes après le début de la présentation de la requête (définie par l'objet `SO_Query_Membres`), une musique est déclenchée.

L'action `Action8` déclenche la musique de fond. L'action déclenchante `Action9` correspond au début de la présentation de la requête, la durée `duration` donne la valeur en secondes au bout de laquelle l'événement est déclenché après cette action.

- *Evénement d'appel de méthodes*

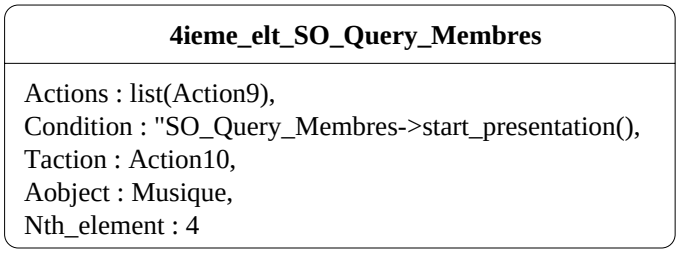

FIG. 5.21 – *Un objet scénario représentant une action suite à un événement d'appel de méthodes*

Un autre type de comportement d'une requête peut être lié à un événement d'appel de méthodes. L'objet `4ieme_elt_SO_Query_Membres` de la classe `Collection_Scenario`, présenté sur la figure 5.21, exprime qu'on veut réaliser une certaine action lorsque le quatrième événement d'appel de la méthode `start_presentation()` se produit après le début de la présentation de la requête.

5.4.3 Interrogation du comportement

Un des grands avantages d'une modélisation objet des événements présentée dans la section 5.4.1 est de pouvoir effectuer des *interrogations sur le comportement d'une présentation.* Tous les comportements sont définis à l'aide d'objets, ils sont référencés dans une liste de comportements à travers l'attribut `behaviours` d'un objet `SO` (`behaviours : set(Scenario)`), ainsi ils sont facilement interrogeables. Prenons quelques exemples de requêtes en langage `OQL` interrogeant le comportement d'une présentation. Les racines de persistance utilisées sont les suivantes :

- `SO_Images : set(SO_Image),`
- `Shows : set(Show),`
- `Scenarios : set(Scenario),`
- `Navigation_Scenarios : set(Navigation_Scenario),`
- `Synch_Scenarios : set(Synch_Scenario)`

1. *R1 : sélectionner toutes les présentations liées à la photo d'Agnès :*

Select *s*
from *s in Shows, Photo_Agnès in SO_Images,*
sce in Photo_Agnès->behaviours, sce in Navigation_Scenarios
where *sce->Aobject = s*

`s` est la présentation cible de l'objet `Scenario sce` décrivant un lien navigationnel à partir de la photo d'Agnès.

2. *R2: sélectionner tous les liens navigationnels de la présentation "*`Show_Membres`*":*

Select *sce*
from *sce in Navigation_Scenarios, s in Shows, so in s->components*
where *s->content->name = "Show_membres" and*
sce in so->behaviours

3. *R3: sélectionner tous les types de synchronisation spécifiques à une présentation et les objets concernés:*

Select *tuple(c: sce->constraint, o1: so, o2: sce->Aobject)*
from *sce in Synch_Scenarios, s in Shows, so in s->components*
where *s->content->name = "Show_Membres" and*
sce in so->behaviours

L'objet `Scenario sce` décrit une contrainte de synchronisation spécifique (`par_finish`) entre l'objet "`so`" (de type `SO`) et l'objet "`e->Aobject`" (qui est aussi de type `SO`).

Les interrogations possibles sont bien sûr multiples et seront différentes suivant les besoins de l'utilisateur.

Les requêtes que nous venons de décrire nécessitent de bien connaître le schéma, ce qui n'est pas toujours le cas. En considérant la complexité de la structure des données, nous proposons certaines extensions comprenant des expressions de type GPE (Generalized Path Expressions) qui ont été initialement proposées dans le contexte de documents [CACS94]. Nous utilisons GPE pour pouvoir dissimuler la structure de notre schéma et ainsi pouvoir définir clairement et facilement différents types de requêtes. Nous constatons en effet que la complexité des requêtes a diminué :

1. *R1: sélectionner toutes les présentations liées à la photo d'Agnès:*

Select *s*
from *s in Shows, SO{Photo_Agnès}, Navigation_Scenarios{S}*
where *S.Aobject = s and*
S in Photo_Agnès->behaviours

2. *R2 : sélectionner tous les liens navigationnels de la présentation "*`Show_Membres`*" :*

Select *S*
from *Navigation_Scenarios{S}, Shows{s}..name, so in s->components*
where *name = "Show_Membres" and*
S in so->behaviours

3. *R3 : sélectionner tous les types de synchronisation spécifiques à une présentation et les objets concernés :*

Select *tuple(c: S->constraint, o1: so, o2: S->Aobject)*
from *Synch_Scenarios{S}, s in Shows, so in s->components*
where *s->content->name = "Show_Membres" and*
S in so->behaviours

5.4.4 Exécution des comportements à l'aide de règles actives

Nous avons étudié les différents types de comportements se produisant au cours d'une présentation et les effets qu'ils pouvaient engendrer. Ces comportements sont stockés sous forme d'objets. Il faut maintenant décrire la façon dont ils seront joués au cours du déroulement de la présentation.

Nous avons constaté que la sémantique des objets scénarios est très proche de celle d'une règle active E-C-A (Evénement-Condition-Action). En effet, ces objets comprennent la définition de l'événement déclencheur, les actions déclenchées suivant une certaine condition et les différents objets concernés par ce comportement. Or une règle active a la sémantique suivante : *lorsqu'un événement se produit et si la condition est satisfaite alors exécuter l'action.* Les règles actives permettent de réagir à un événement et d'exécuter différentes actions suivant une certaine condition. Ainsi, les différents *objets scénarios* sont traduits en règles actives pour pouvoir être joués au cours de la présentation.

Dans un premier temps, on montre qu'elles permettent de respecter les contraintes de synchronisation et, dans un deuxième temps, comment elles peuvent permettre

l'exécution des différents comportements associés aux objets d'une présentation. Nous nous appuierons sur les travaux sur les SGBD actifs [CKAK94, CC96c] et plus particulièrement sur le système NAOS [CCS94].

Remarque - L'exécution d'une présentation pourrait se faire seulement à l'aide de règles actives. En effet, chaque changement dans une présentation correspond à un événement (*début de la présentation d'un objet*, *fin de la présentation d'un objet*, etc.). Mais nous ne voulons pas modifier notre moteur d'exécution actuel de présentations multimédias, nous choisissons d'utiliser les règles actives seulement pour des événements bien spécifiques.

5.4.4.1 Les événements déclencheurs de règles actives

Les travaux sur les SGBD actifs distinguent deux grandes catégories d'événements, les événements primitifs et composites. Les événements composites sont composés de plusieurs événements primitifs à l'aide de différents opérateurs (*séquence*, *disjonction*, *conjonction*, etc.).

- **Les événements primitifs**

  Les événements primitifs, définis dans une base de données active comme NAOS, qui sont susceptibles de déclencher des scénarios sont : des *événements utilisateurs*, des *événements temporels* et des *événements de manipulation d'entités* [Fay97].

  1. **Evénements utilisateurs**

     Les *événements utilisateurs* sont des événements déclenchés explicitement par l'utilisateur au sens large du terme (programme, fonction, etc.). Le développeur peut les utiliser pour signaler un événement particulier lié au contexte de l'application. Ils font partie des types d'événements externes.

  2. **Evénements temporels**

     Les *événements temporels* peuvent être divisés en deux grandes catégories :

     - les événements temporels absolus;
     - les événements temporels périodiques.

     Un exemple d'événement temporel absolu serait : *Le 8 mai 1997 à 15h30.* Une règle basée sur un événement temporel absolu ne s'exécute donc qu'une seule fois. Un exemple d'événement temporel périodique serait : *Tous les*

mois. Une règle basée sur un événement temporel périodique s'exécute plusieurs fois. Pour en limiter le nombre, on peut utiliser un intervalle de temps défini par deux valeurs temporelles absolues. De plus, les événements temporels peuvent être subordonnés à un autre événement. En combinant cette possibilité aux deux types d'événements temporels vus précédemment, deux nouveaux types d'événements apparaissent :

- les événements temporels absolus relatifs;
- les événements temporels périodiques relatifs.

Un exemple d'événement temporel absolu relatif serait : *2 heures après e1.* Un exemple d'événement temporel périodique relatif serait : *Tous les jours après e2.*

3. **Evénements de manipulations d'entités**

   Ces événements sont liés aux manipulations de données suivantes : ajout, recherche, suppression, modification, exécution d'une méthode. Dans une présentation, ce qui nous intéresse surtout, sont les événements d'appel de méthodes (par exemple, la méthode `start_presentation` sur les objets `SO`).

- **Les événements composites**

Un type d'événement composite est défini par une expression incluant des types d'événements primitifs ou composites et des opérateurs de composition dont les plus répandus sont la *disjonction*, la *conjonction* et la *séquence* [CKAK94, CC96c]. Nous utilisons, pour la définition de nos règles actives, deux types d'opérateurs de composition dont nous rappelons la définition :

Soit {E1, E2, ... , En} des types d'événements et {e1, e2, ... , en} leurs instances respectives.

Séquence - Un type d'événement caractérisant la séquence de deux événements e1 et e2 est de la forme : E1 , E2. Une instance de ce type se produit si le **dernier** événement qui compose e1 se produit avant le **dernier** événement qui compose e2.

Négation - Un type d'événement caractérisant la négation d'un événement e1 est de la forme : !E1. Une instance de ce type se produit si e1 ne se produit pas dans un intervalle de validité.

5.4.4.2 Exécution des délais et durées interactifs

- **Délai interactif**

Un délai interactif correspond à un intervalle durant lequel l'utilisateur peut interagir pour démarrer la présentation. L'exécution du délai interactif lié à l'objet `Musique_Mozart` (c.f. Figure 4.9), dont la valeur est l'intervalle [30s, 1mn], se fait à l'aide des trois règles actives dont la sémantique est la suivante :

(1) *30s après le début de Musique_Mozart,*
activer le bouton Tobject1.
(2) *clic sur le bouton Tobject1,*
jouer Musique_Mozart.
(3) *60s après le début de Musique_Mozart,*
désactiver le bouton Tobject1.

La première règle va activer le bouton Tobject1 30 secondes après le début de Musique_Mozart. La deuxième règle va réagir à l'action de l'utilisateur. Si celui-ci clique sur le bouton `Tobject1` alors la musique commence. Enfin, la dernière règle désactive le bouton au bout d'une minute de présentation de Musique_Mozart. Ces règles assurent que l'interaction est effectivement possible dans l'intervalle [30s, 1mn] après le début de Musique_Mozart.

Formalisme NAOS - Dans le formalisme proposé par le système NAOS [CCS94], ces règles actives sont définies de la manière suivante :

(1) CREATE RULE Delay_Musique_Mozart1
COUPLING IMMEDIATE
ON TEMPORAL EVENT 30s AFTER METHOD_BEGIN
SO_Audio->start_presentation WITH m1
IF (m1->content->name = "Musique_Mozart" and
d1 in m1->behaviours and d1->name = "Delay_Musique_Mozart")
DO { d1->Tobject->activate(); }

Pour une question de simplification, nous ne donnerons pas un nom à chacune des règles actives qui suivent et ne définirons pas non plus le mode de couplage qui est toujours immédiat. En effet, toutes nos règles actives sont déclenchées dès que l'événement se produit.

(2) ON USER EVENT clic(Tobject1 : Object, m1 : SO_Audio)
IF (m1->content->name = "Musique_Mozart" and
d1 in m1->behaviours and d1->Tobject = Tobject1)
DO { m1->play(); }

(3) ON TEMPORAL EVENT 60s AFTER METHOD_BEGIN
SO_Audio->start_presentation WITH m1
IF (m1->content->name = "Musique_Mozart" and
d1 in m1->behaviours and d1->name = "Delay_Musique_Mozart")
DO { d1->Tobject->desactivate(); }

- **Durée interactive**

Une durée interactive correspond à un intervalle durant lequel l'utilisateur peut interagir pour stopper la présentation. L'exécution de la durée interactive liée à l'objet `Photo_thésards` (c.f. Figure 4.10), dont la valeur est l'intervalle [10s, 30s], se fait à l'aide de trois règles actives dont la sémantique est la suivante :

(1) 10s après l'affichage de Photo_thésards,
activer le bouton Tobject1.
(2) clic sur le bouton Tobject1,
désafficher Photo_thésards.
(3) 30s après l'affichage de Photo_thésards,
désactiver le bouton Tobject1.

L'interaction est seulement possible dans l'intervalle [10s, 30s]. Par un clic souris sur le bouton Tobject1, l'utilisateur peut arrêter la présentation de l'image Photo_thésards.

Formalisme NAOS -

(1) ON TEMPORAL EVENT 10s AFTER METHOD_BEGIN
SO_Image->play WITH i1
IF (i1->content->name = "Photo_thésards" and
d1 in i1->behaviours and d1->name = "Duration_Photo_thésards")
DO { d1->Tobject->activate(); }

(2) ON USER EVENT clic(Tobject1 : Object, i1 : SO_Image)
IF (i1->content->name = "Photo_thésards" and

d1 in i1->behaviours and d1->Tobject = Tobject1)
DO { i1->end_presentation(); }

(3) ON TEMPORAL EVENT 30s AFTER METHOD_BEGIN
SO_Image->play WITH i1
IF (i1->content->name = "Photo_thésards" and
d1 in i1->behaviours and d1->name = "Duration_Photo_thésards")
DO { d1->Tobject->desactivate(); }

5.4.4.3 Respect des contraintes de synchronisation

Dans le paragraphe 4.3.2, nous avons défini la sémantique d'un certain nombre de contraintes. Nous allons montrer comment utiliser les règles actives pour le respect de ces contraintes de synchronisation, elles deviennent une aide dans l'exécution correcte d'une présentation. En effet, elles vont imposer le respect des contraintes suivant les relations de causalité définies. Les contraintes de synchronisation sont appliquées au parallélisme ou à la séquentialité.

1. Contraintes de synchronisation parallèles

- **Contrainte par_equal**

  Prenons l'exemple de la contrainte de synchronisation `par_equal` sur deux objets `o1` et `o2` comme nous la décrit la figure 5.22. Ces deux objets `SO` possèdent une *Ombre Temporelle*.

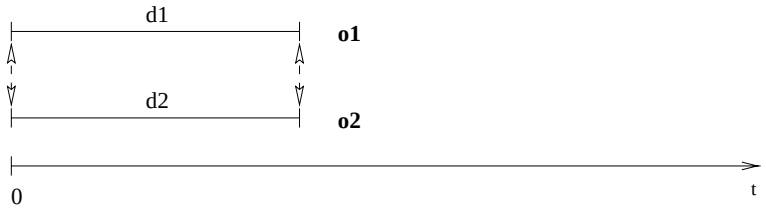

FIG. 5.22 – *Présentation parallèle de deux objets o1 et o2.*

La contrainte `par_equal` impose que les présentations des deux objets commencent en même temps et se terminent aussi en même temps lorsque l'une d'entre elles s'arrête.

Les deux règles actives dont la sémantique est la suivante permettent d'assurer que `o1` et `o2` commencent en même temps :

R1 : *Si le début de o2 n'a pas lieu avant le début de o1, démarrer o2.*
R2 : *Si le début de o1 n'a pas lieu avant le début de o2, démarrer o1.*

Formalisme NAOS -

R1 : ON !METHOD_BEGIN SO->start_presentation WITH o2 ,
METHOD_BEGIN SO->start_presentation WITH o1
IF (o1->content->name = "o1" and o2->content->name = "o2")
DO { o2->start_presentation(); }

R2 : ON !METHOD_BEGIN SO->start_presentation WITH o1 ,
METHOD_BEGIN SO->start_presentation WITH o2
IF (o1->content->name = "o1" and o2->content->name = "o2")
DO { o1->start_presentation(); }

Suivant l'événement qui se produit, *o1->start_presentation* ou *o2->start_presentation*, une des deux règles actives est déclenchée. Ainsi on permet le cas où la présentation de l'objet `o1` commence en premier et démarre celle de `o2` ou le contraire.

Remarque - Ces deux règles actives peuvent être utilisées dans le cas d'une contrainte de synchronisation `par_start` entre les objets `o1` et `o2`.

Les deux prochaines règles actives assurent que les présentations des objets `o1` et `o2` se terminent en même temps. Soit la fin de la présentation de `o1` stoppe celle de `o2` si celle-ci se produit en premier, soit le contraire.

R3 : ON !METHOD_END SO->end_presentation WITH o2 ,
METHOD_END SO->end_presentation WITH o1
IF (o1->content->name = "o1" and o2->content->name = "o2")
DO { o2->end_presentation(); }

R4 : ON !METHOD_END SO->end_presentation WITH o1 ,
METHOD_END SO->end_presentation WITH o2

IF (o1->content->name = "o1" and o2->content->name = "o2")
DO { o1->end_presentation(); }

Remarque - Ces deux règles actives peuvent être utilisées dans le cas d'une contrainte de synchronisation `par_finish` entre les objets `o1` et `o2`.

Problème - Nous considérons dans cet exemple que `par_equal(o1, o2)` forme une présentation dont les objets `o1` et `o2` sont uniques. Les règles actives définies précédemment posent un problème si le même objet `SO` est utilisé plusieurs fois dans une présentation, et qu'il est lié à d'autres objets suivant différentes contraintes de synchronisation. Toutes les règles actives définies pour vérifier ces contraintes de synchronisation ne doivent pas être déclenchées au même moment au cours du déroulement de la présentation.

Ainsi il est nécessaire de définir d'autres règles actives qui seront liées cette fois-ci au *Show* définissant la contrainte de synchronisation. Par exemple, si la contrainte `par_equal(o1, o2)` fait partie d'une présentation plus complexe, alors il existe un objet `par_equal1` (représentant un noeud de l'arbre de la présentation) la définissant. Nous définissons les règles suivantes pour rendre actives les règles vérifiant la contrainte au début de la présentation de `par_equal1` et inactives à la fin de sa présentation :

R5 : ON METHOD_BEGIN Par_equal->start_presentation WITH p1
IF (p1 = par_equal1)
DO { R1->enable; R2->enable;
R3->enable; R4->enable; }

R6 : ON METHOD_END Par_equal->end_presentation WITH p1
IF (p1 = par_equal1)
DO { R1->disable; R2->disable;
R3->disable; R4->disable; }

- **Contrainte par_during**

La contrainte de synchronisation `par_during(o1, o2)` impose que la présentation de l'objet `o1` commence avant celle de `o2` et qu'elle finisse après la fin de la présentation de `o2`.

Les deux règles actives suivantes permettent d'assurer la contrainte de synchronisation `par_during(o1, o2)` :

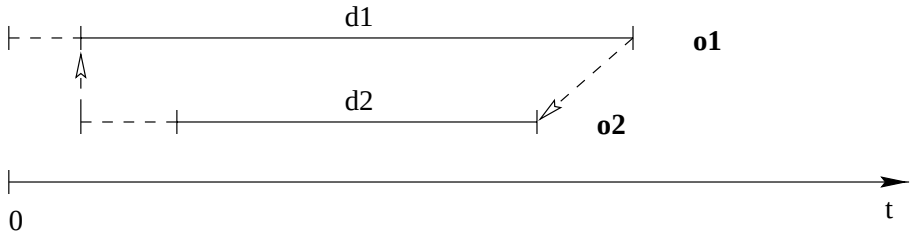

FIG. 5.23 – *Présentation par_during de deux objets o1 et o2.*

```
ON !METHOD_BEGIN SO->start_presentation WITH o1 ,
    METHOD_BEGIN SO->start_presentation WITH o2
IF (o1->content->name = "o1" and o2->content->name = "o2")
DO { o1->play(); }

ON !METHOD_END SO->end_presentation WITH o2 ,
    METHOD_END SO->end_presentation WITH o1
IF (o1->content->name = "o1" and o2->content->name = "o2")
DO { o2->end_presentation(); }
```

La première règle active impose qu'au début de la présentation de `o2` (événement *o2->start_presentation*), si la présentation de `o1` n'a pas démarré (l'événement *o1->start_presentation* ne s'est pas produit), alors on déclenche l'observation de `o1`. Ainsi on ne tient plus compte du délai de `o1`, puisque la présentation de `o2` aurait du débuter à la fin de ce délai.

La deuxième règle stoppe la présentation de `o2` si elle n'a pas eu lieu avant la fin de la présentation de `o1`.

- **Contrainte par_overlap**

La contrainte de synchronisation `par_overlap(o1, o2)` impose que la présentation de l'objet `o1` commence et finisse avant celle de `o2`.

Les deux règles actives suivantes permettent d'assurer la contrainte de synchronisation `par_overlap(o1, o2)`, la première déclenche l'observation de `o1` au début de la présentation de `o2` et la deuxième stoppe la présentation de `o1` si elle n'a pas eu lieu avant la fin de la présentation de `o2`:

```
ON !METHOD_BEGIN SO->start_presentation WITH o1 ,
    METHOD_BEGIN SO->start_presentation WITH o2
```

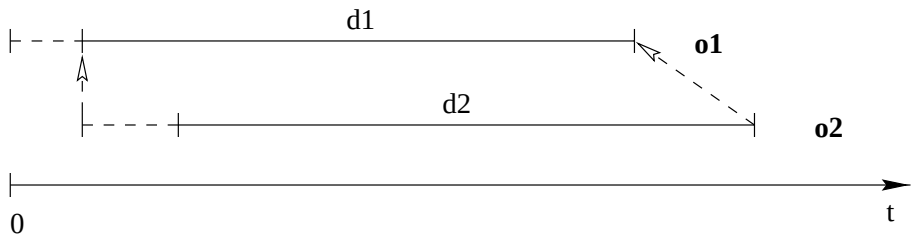

FIG. 5.24 – *Présentation par_overlap de deux objets o1 et o2.*

IF (o1->content->name = "o1" and o2->content->name = "o2")
DO { o1->play(); }

ON !METHOD_BEGIN SO->end_presentation WITH o1 ,
METHOD_BEGIN SO->end_presentation WITH o2
IF (o1->content->name = "o1" and o2->content->name = "o2")
DO { o1->end_presentation(); }

2. Contraintes de synchronisation séquentielles

- **Contrainte seq_before**

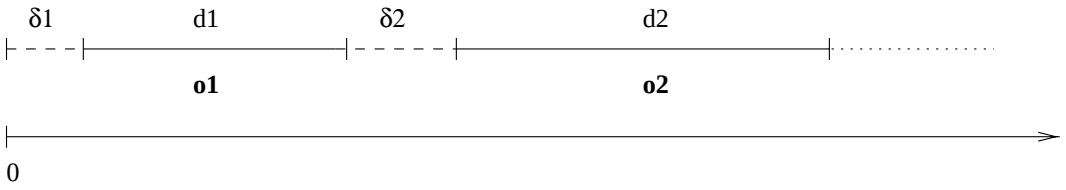

FIG. 5.25 – *Présentation séquentielle de deux objets o1 et o2.*

Cet exemple montre une présentation séquentielle de deux objets `o1` et `o2`. Ils sont présentés au niveau de l'interface en séquence. Le premier est présenté après un délai $\delta 1$ et pendant une durée `d1`, on attend alors un délai $\delta 2$ pour présenter `o2` pendant une durée `d2`.

La règle active suivante permet la synchronisation entre les deux objets de la présentation, dans le cas où la fin de la présentation de `o1` est survenue (événement *o1->end_presentation*) et que celle de `o2` n'a pas démarré après un délai $\delta 2$ (l'événement *o2->start_presentation* ne s'est pas produit). La contrainte de

synchronisation étant `seq_before`, le deuxième objet doit être joué.

```
ON !METHOD_BEGIN SO->start_presentation WITH o2 ,
    TEMPORAL EVENT δ2 AFTER METHOD_END SO->end_presentation WITH o1
IF (o1->content->name = "o1" and o2->content->name = "o2")
DO { o2->start_presentation; }
```

- **Contrainte seq_meet**

  Dans l'exemple de la figure 3.12, la contrainte de synchronisation entre la présentation des images est `seq_meet`, elle impose un délai nul entre chaque présentation d'image. Plusieurs règles actives peuvent être alors associées à la séquence d'images, chacune vérifiant la contrainte de synchronisation :

```
ON !METHOD_BEGIN SO_Image->start_presentation WITH I2 ,
    METHOD_END SO_Image->end_presentation WITH I1
IF (I1->content->name = "I1" and I2->content->name = "I2")
DO { I2->start_presentation; }

ON !METHOD_BEGIN SO_Image->start_presentation WITH I3 ,
    METHOD_END SO_Image->end_presentation WITH I2
IF (I3->content->name = "I3" and I2->content->name = "I2")
DO { I3->start_presentation; }

ON !METHOD_BEGIN SO_Image->start_presentation WITH I4 ,
    METHOD_END SO_Image->end_presentation WITH I3
IF (I3->content->name = "I3" and I4->content->name = "I4")
DO { I4->start_presentation; }
```

5.4.4.4 Exécution des différents comportements

Une règle active peut être déclenchée par les différents types d'événements qui déclenchent une certaine action correspondant à un certain type de comportements. L'événement déclencheur ne se produira pas à chaque exécution de la présentation suivant le scénario choisi pour celle-ci.

Nous donnons ici deux exemples d'exécution de comportements à l'aide de règles actives. Bien sûr, tous les comportements décrits au cours du chapitre 4 sont réalisables

à l'aide de règles actives.

1. Réalisation d'un lien navigationnel

   Les règles actives vont nous permettre de réaliser les liens navigationnels entre deux présentations multimédias. Reprenons l'exemple de la figure 4.12 qui permet de lier deux présentations multimédias.

   Nous avons décrit au paragraphe 5.4.2 l'objet "`Navigation_Show_Agnès`" créé pour définir ce comportement. Nous allons étudier maintenant comment il peut être exécuté à l'aide de règles actives. La règle active dont la sémantique est la suivante est exécutée si l'utilisateur clique sur la photo d'Agnès :

   > *Le début de la présentation Photo_Agnès,*
   > *suivi de l'activation du bouton Photo_Agnès_Choice,*
   > *suivi d'un clic souris sur ce bouton,*
   > *entraîne la fin de la présentation de Show_Membres*
   > *et le début de la présentation de Show_Agnès.*

   Formalisme NAOS -

```
ON METHOD_BEGIN SO_Image->start_presentation WITH p1 ,
    METHOD_BEGIN Object->activate WITH c1 ,
    USER EVENT clic(Photo: SO_Image)
IF (p1->content->name = "Photo_Agnès" and c1->name = "Photo_Agnès_choice"
    and p1 = Photo and n1 in p1->behaviours
    and n1->name = "Navigation_Show_Agnès"
    and p1 in s1->components and s1->content->name = "Show_Membres")
DO { s1->end_presentation();
     n1->Aobject->start_presentation(); }
```

   Dans le cas où la présentation principale n'est pas stoppée, mais seulement arrêtée pendant le temps d'exécution de l'autre présentation, le comportement est légèrement différent et exécuté à l'aide des deux règles actives suivantes :

```
ON METHOD_BEGIN SO_Image->start_presentation WITH p1 ,
    METHOD_BEGIN Object->activate WITH c1 ,
    USER EVENT clic(Photo: SO_Image)
IF (p1->content->name = "Photo_Agnès" and c1->name = "Photo_Agnès_choice"
```

```
    and p1 = Photo and n1 in p1->behaviours
    and n1->name = "Navigation_Show_Agnès"
    and p1 in s1->components and s1->content->name = "Show_Membres")
DO { s1->resume();
     n1->Aobject->start_presentation(); }
```

La première exécute le comportement, elle fait une pause sur la présentation `Show_Membres` et démarre `Show_Agnès`. Et la deuxième redémarre `Show_Membres` à la fin de `Show_Agnès`.

2. Réalisation d'une contrainte de synchronisation spécifique

Les deux règles actives suivantes réalisent la contrainte de synchronisation spécifique `par_finish` entre les objets `Texte_PresentationProjet` et `Image_Chef` de la figure 4.13 :

```
ON !METHOD_BEGIN SO_Image->end_presentation WITH i1 ,
    METHOD_BEGIN SO_Text->end_presentation WITH t1 ,
IF (i1->content->name = "Image_Chef"
    and t1->name = "Texte_PresentationProjet"
    and n1 in i1->behaviours and n2 in t1->behaviours
    and n1->name = "Par_finish_IC"
    and n2->name = "Par_finish_TPP")
DO { i1->end_presentation; }

ON !METHOD_BEGIN SO_Image->end_presentation WITH t1 ,
    METHOD_BEGIN SO_Text->end_presentation WITH i1 ,
IF (i1->content->name = "Image_Chef"
    and t1->name = "Texte_PresentationProjet"
    and n1 in i1->behaviours and n2 in t1->behaviours
    and n1->name = "Par_finish_IC"
    and n2->name = "Par_finish_TPP")
DO { t1->end_presentation; }
```

5.4.4.5 Intégration au moteur d'exécution de présentations

L'exécution du déroulement de la présentation n'est pas modifiée mais il faut détecter un certain nombre d'événements pendant le temps de la présentation qui vont dé-

clencher l'exécution de règles actives. Or le *détecteur d'événements* du système NAOS est capable de détecter tous les types d'événements que nous avons définis.

Au début de chaque présentation d'un objet `SO`, un certain nombre de règles actives sont activées à l'aide de la méthode `start_behaviour` de la classe `SO` décrite précédemment. Ces règles actives correspondent aux différents comportements définis sur cet objet. Elles sont donc prêtes à réagir à différents événements au cours de la présentation. La méthode `start_behaviour` va aussi réaliser un pré-chargement des objets `SO` externes à cette présentation mais dont les présentations peuvent être déclenchées au cours de la présentation suivant la définition des comportements de celle-ci. Nous connaissons tous ces objets `SO` externes grâce à la modélisation objet des comportements, ils sont référencés par les *objets scénarios* liés à la présentation ou à ces objets composants.

5.4.5 Création des comportements

La **bibliothèque de classes de comportements** propose un ensemble de comportements prédéfinis. Nous allons définir l'interface permettant d'associer différents comportements à une présentation et ainsi de définir son *Ombre Comportementale*. Une fois définis par l'utilisateur, les comportements sont stockés sous forme d'*objets scénarios*.

5.4.5.1 Interface de saisie à l'aide de menus et écrans

L'interface que nous présentons tout d'abord se base sur différents écrans et menus comme pour la création d'une présentation multimédia. Au moment de la saisie des objets `SO` qui composent la présentation, des menus présentant les comportements sont proposés aux utilisateurs. Nous allons décrire les menus et les écrans de saisie qui permettent de créer des objets scénarios :

- Après saisie d'un objet `SO` de la présentation

  Après la création d'un objet `SO` (de type `SO_X` où X = Image|Texte|Audio|Video), l'utilisateur a à sa disposition un menu des différents comportements possibles qu'il peut associer à cet objet, dont entre autres un lien navigationnel.

  Si l'utilisateur choisit l'option "**lien navigationnel**", un menu de définition d'un lien navigationnel apparaît avec trois options "*objet cible*", "*pause*" et "*arrêt*" qui correspondent à :

- l'*objet cible* (la présentation qui est liée à cet objet, dans l'exemple du paragraphe 4.4.1, il s'agit de `Show_Agnès`) : l'utilisateur clique sur cette option et il a alors deux possibilités : (1) soit choisir un Show existant à partir d'une liste des shows de la base (puisqu'il s'agit d'une présentation indépendante sous forme d'un objet `SO`, il peut la jouer pour la choisir); (2) soit en créer un nouveau.

- le *type de l'action* : l'utilisateur a le choix entre les deux options suivantes soit "`pause`", soit "`arrêt`" (ce choix détermine si la présentation principale `Show_membres` est momentanément interrompue ou stoppée définitivement). Il suffit qu'il en sélectionne une des deux. Ensuite, il valide et l'objet scénario représentant le lien navigationnel est créé et lié à l'objet `SO` qui venait d'être créé.

- Après saisie de toute la présentation

Pour tout ce qui concerne les comportements liés à la présentation, ceux-ci sont définis à la fin de la création de la présentation. Une fois que l'utilisateur a construit sa présentation et avant sa validation finale, il peut jouer sa présentation pour voir si elle lui convient, mais il peut aussi lui associer un certain nombre de comportements à l'aide d'un menu.

Dans ce menu, il peut choisir "**comportement temporel**" pour définir un comportement lié à un événement temporel. Une fenêtre apparaît lui demandant certaines données spécifiant ce comportement :

- la *durée* : il s'agit du temps en secondes où l'événement est déclenché par rapport au temps de début de la présentation.

- l'*objet concerné* : il s'agit de l'objet `SO` sur lequel une méthode sera appliquée (cette méthode étant définie par l'action choisie). L'utilisateur pourra au choix créer cet objet ou le choisir dans une liste d'objets de la base.

- l'*action déclenchée* : il s'agit de l'appel d'une méthode sur un objet `SO` (par exemple, la méthode `start_presentation` pour lancer la présentation de cet objet). Un certain nombre d'actions est proposé à l'utilisateur comme "jouer la présentation de l'objet" (méthode `start_presentation`), "observer l'objet" (méthode `play`), etc.

A ce niveau de l'interface, nous allons aussi permettre la création de contraintes de synchronisation spécifiques en choisissant dans le menu des différents comportements l'option "**contrainte de synchronisation spécifique**". En effet, ce type de comportement concerne deux objets de la présentation, il faut ainsi pour le créer avoir saisi tous les objets `SO` qui la composent. Une fenêtre apparaît demandant à l'utilisateur de définir :

- le *type de contrainte* : il a le choix entre tous les types de synchronisation (`par_start`, `par_equal`, `seq_meet`, etc), il choisit pour l'exemple la contrainte "`par_finish`".

-le *premier objet* `SO` : il va être lié au deuxième objet `SO` choisi par la contrainte `par_finish`. L'utilisateur peut créer cet objet ou le sélectionner parmi une liste des objets existants.

- le *deuxième objet* `SO` : il s'agit de l'autre objet qui est lié par la contrainte de synchronisation spécifique au premier objet `SO` choisi (dans l'exemple du paragraphe 4.4.2, les deux objets `SO` "`Texte_PrésentationProjet`" et "`Image_Chef`" sont liés par la contrainte de synchronisation `par_finish`).

- Après saisie d'une requête de la présentation

Un menu apparaît après création d'un objet `SO` dont le contenu est une requête, il propose à l'utilisateur différents comportements possibles. Si il choisit l'option "**comportement temporel**", une fenêtre apparaît lui demandant :

- la *durée* : il s'agit du temps en secondes où l'événement est déclenché par rapport au temps de début de la présentation de la requête.

- l'*objet concerné* : il s'agit de l'objet `SO` sur lequel une méthode sera appliquée (action choisie).

- l'*action déclenchée* : il s'agit de l'appel d'une méthode sur l'objet `SO` choisi.

Si il chosisit l'option "**comportement lié à un appel de méthodes**" dans le menu des comportements, une fenêtre apparaît demandant à l'utilisateur :

- le *Nième élément* : il s'agit du numéro de l'élément auquel est lié le comportement.

- l'*action déclenchée* : il s'agit de l'appel de la méthode `start_presentation` sur l'objet `SO`, *Nième élément* du résultat de la requête.

Remarque - Une liste de scénarios déjà existants sera toujours proposée à l'utilisateur, dans un but de réutilisabilité. Il s'agit des scénarios qui ont été créés par d'autres auteurs de présentations dans le contexte de l'application ou d'autres suivant les besoins.

5.4.5.2 Interface graphique

Nos pourrions aussi utiliser notre atelier graphique pour la création des comportements (celui-ci est décrit en annexe A). La figure 5.27 présente quelques icones permettant de définir certains comportements. Le premier icone de la première ligne représente une action de l'utilisateur, ensuite le deuxième correspond à un appel de méthodes et enfin le troisième à un événement temporel. L'icone "lien navigationnel" représente tout simplement un lien vers une autre présentation.

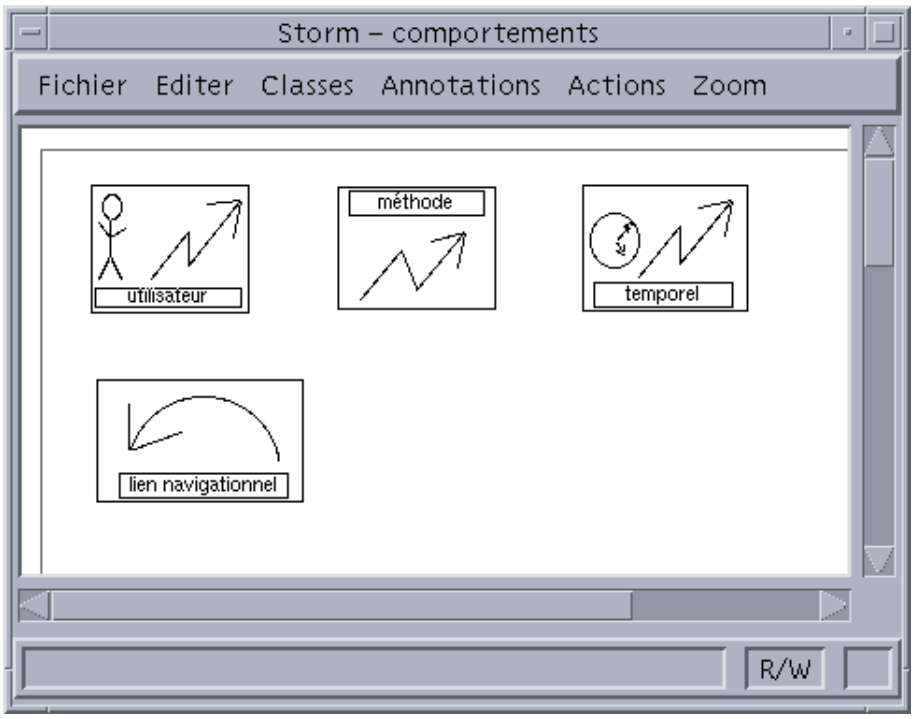

FIG. 5.26 – *Icones correspondant aux comportements proposés*

La figure 5.27 présente un exemple de présentation avec un lien navigationnel au niveau de l'image "`Photo_Agnès`" vers une autre présentation "`Show_Agnès`. La représentation graphique du lien navigationnel nous permet une visualisation des différents comportements qui sont liés au déroulement de la présentation. Après validation de la présentation et au moment de sa création sous forme d'un objet `SO`, les différents objets scénarios correspondant aux différents comportements sont aussi créés.

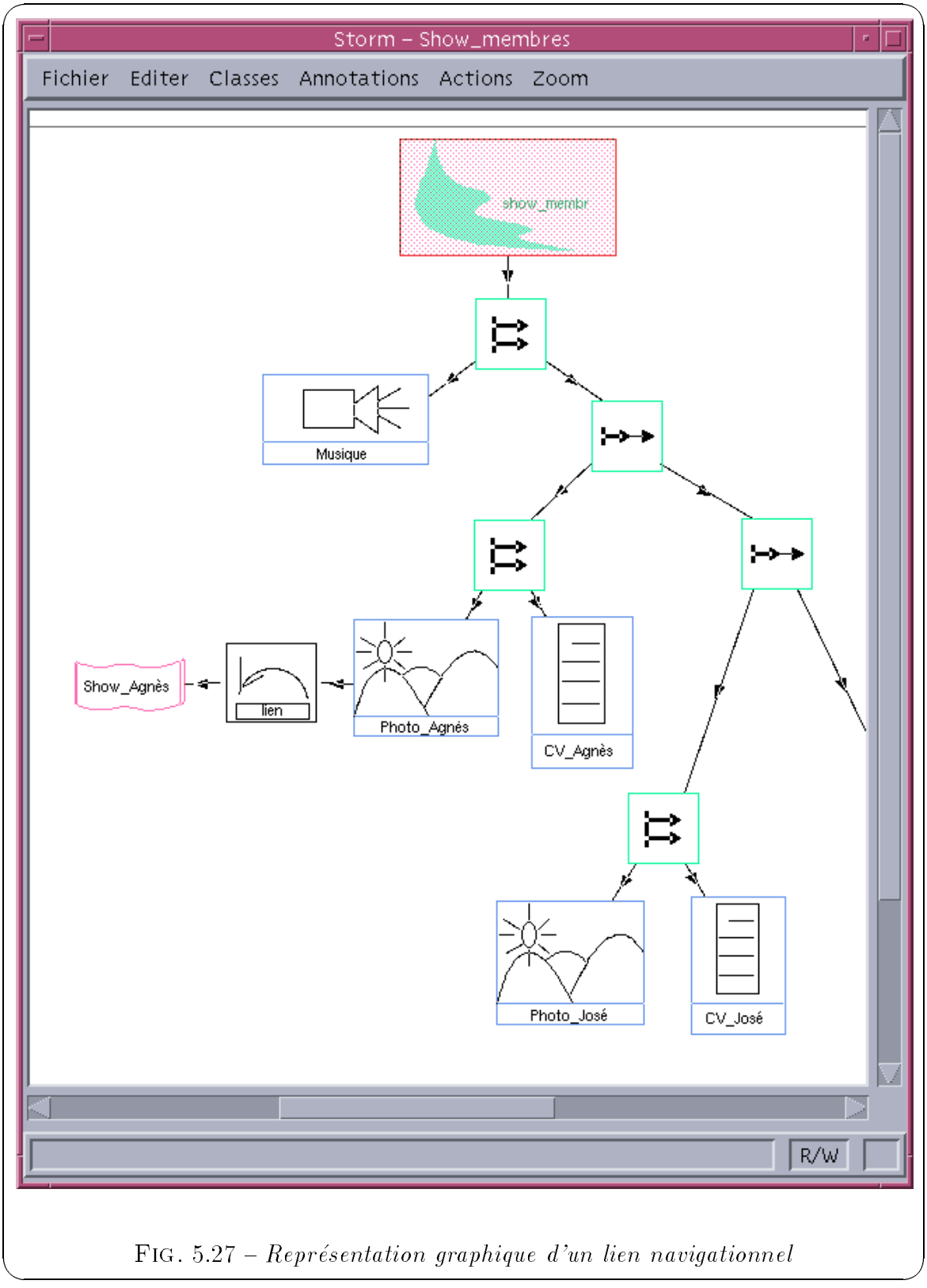

FIG. 5.27 – *Représentation graphique d'un lien navigationnel*

5.5 Conclusion

Dans ce chapitre, nous avons décrit notre implantation d'une extension à un SGBD à objets dans le but de pouvoir manipuler et gérer des données multimédias telles que l'image, le texte, le son et la vidéo, mais aussi des présentations multimédias dans le but de proposer des fonctionnalités pour construire des applications multimédias. L'architecture de notre SGBD multimédia s'appuie sur celle d'un serveur d'objets classique, nous ajoutons une couche supplémentaire qui correspond à un gestionnaire de présentations, il permet de construire, jouer et interroger les objets multimédias. L'interface utilisateur comporte certaines adaptations pour l'édition, la synchronisation et l'interrogation des objets multimédias.

Les extensions multimédias aux SGBD à objets que nous proposons sont modélisées sous forme de différents objets. Nous avons défini tout d'abord les objets multimédias qui permettent de stocker les données multimédias dans la base mais aussi de les gérer. Ensuite, nous avons un ensemble d'objets correspondant à diverses présentations des objets multimédias de la base et dont les classes appartiennent à la **bibliothèque de classes STORM**. Ces objets possèdent des aspects temporels sous forme d'une *Ombre Temporelle*, ils peuvent être synchronisés entre eux en séquence ou en parallèle pour former des présentations plus complexes. Le principal avantage de pouvoir stocker ces présentations sous forme d'objets dans la base de données est de pouvoir les rechercher facilement à l'aide d'un langage de requêtes pour les rejouer par exemple. Le langage OQL avec certaines adaptations nous permet d'interroger les présentations sur les aspects descriptifs comme leurs mot-clés, mais aussi sur les aspects temporels comme leur durée. Enfin, il est aussi possible de les interroger sur la synchronisation entre les objets qui les composent. Toutes ces possibilités d'interrogation sont réalisables puisque nous nous situons dans le cadre d'un SGBD.

Enfin, nous avons proposé notre implantation de l'*Ombre Comportementale* sous forme d'un ensemble d'*objets scénarios* permettant d'associer différents somportements à une présentation. Ces objets sont des instances des classes appartenant à notre **bibliothèque de classes de comportements** proposé à l'utilisateur. Et nous montrons comment nous pouvons exécuter ces comportements à l'aide de règles actives pour le respect des contraintes de synchronisation et l'exécution des différents comportements. Les objets scénarios sont traduits en règles actives pour pouvoir être joués au cours de la présentation. Nous utilisons la technologie des SGBD actifs pour développer notre SGBD. En conclusion, nous avons proposé un SGBD multimédia permettant de construire, jouer et interroger toutes sortes de présentations multimédias.

Chapitre 6

Bilan et perspectives

6.1 Bilan et contributions

Au cours de cette thèse, nous avons abordé l'intégration du multimédia dans les bases de données. Nous cherchons à répondre aux besoins des utilisateurs pour le développement d'applications multimédias. De nombreux secteurs utilisent le multimédia dans leurs applications. Il s'agit de la bureautique, la médecine, la géographie, l'éducation, la formation continue, la météorologie, les banques, les agences de voyages, la publicité, le courrier électronique, la CAO, la CFAO, les ventes sur catalogues électroniques, etc. Les domaines d'applications sont nombreux et le multimédia va devenir essentiel dans de nombreux secteurs de l'informatique.

Les utilisateurs souhaitent utiliser des systèmes simples et conviviaux manipulant ces nouveaux types de données. Il s'agit de données très riches dans le sens où elles véhiculent une grande quantité d'informations. Il faut ainsi développer des *systèmes capables de manipuler à la fois des données traditionnelles et des données multimédias*. Les systèmes auteurs très présents sur le marché actuellement ne peuvent pas offrir cette fonctionnalité, alors que les futurs SGBD seront capables de gérer toutes ces données sans distinction. Les SGBD actuels nécessitent des adaptations pour la manipulation des données multimédias, ils ont besoin de nouveaux outils pour le stockage, l'édition ou l'interrogation des données multimédias. La technologie des SGBD à objets est bien adaptée à la manipulation de ces données, mais comme nous l'avons montré certaines extensions sont néanmoins nécessaires.

A l'heure actuelle, les besoins des bases de données multimédias se répartissent en

cinq grands thèmes :

- le *stockage des données multimédias* sous forme d'objets dans la base de données pour pouvoir les manipuler : le stockage des données multimédias requiert une grande quantité de ressources en terme de capacité de stockage.
- la *modélisation de la structure des données multimédias* qui nécessite des mécanismes d'indexation et une modélisation sémantique et cohérente des abstractions : la sémantique des données doit être modélisée pour permettre des recherches sur le contenu, ainsi que sur l'information textuelle associée aux données multimédias.
- la *modélisation de présentations multimédias* qui composent des objets multimédias possédant des aspects temporels en spécifiant la synchronisation entre eux, parallèle ou séquentielle.
- l'*interrogation des objets multimédias* sur des aspects descriptifs, temporels ou de synchronisation.
- l'*interaction avec l'utilisateur* : l'utilisateur doit pouvoir interargir au cours du déroulement de la présentation. Ainsi, on obtient différents scénarios possibles pour l'exécution d'une présentation.

Notre travail apporte certaines solutions aux trois derniers thèmes. Nous nous sommes, tout d'abord, intéressés à la **modélisation de présentations multimédias** à l'aide du modèle STORM. Une présentation compose différents médias entre eux à l'aide de relations séquentielles ou parallèles. Chaque média est stocké sous forme d'un objet et on lui associe des aspects temporels pour être présenté à l'utilisateur. Ces aspects sont constitués d'un délai et d'une durée et forment l'*Ombre Temporelle*. Nous pouvons associer à un objet monomédia différentes présentations avec différentes ombres temporelles, ainsi l'objet n'est pas dupliqué. Ensuite, une des présentations de l'objet monomédia peut faire partie d'une présentation plus complexe où elle pourra être synchronisée en parallèle avec un autre objet.

Finalement, toutes ces présentations sont stockées sous forme d'objets `SO` dans la base. Ces objets peuvent appartenir à différentes classes appartenant à une **bibliothèque de classes** prédéfinies et réutilisables que nous proposons à l'utilisateur pour définir ces présentations. Un objet `SO` est constitué d'une *Ombre Temporelle* et d'un contenu. Ce contenu est soit atomique pour la présentation d'un objet monomédia, soit complexe pour la présentation de plusieurs objets monomédias synchronisés en

séquence ou en parallèle. Nous avons ajouté une autre dimension à nos objets SO sous forme d'une *Ombre Comportementale*. Elle associe à la présentation différents comportements.

On appelle *comportement d'une présentation d'un objet*, l'ensemble des actions qu'il est susceptible d'entreprendre suite à des événements lors de sa présentation. En effet, le déroulement de la présentation d'un objet va être influencé par des événements de l'utilisateur ou par des événements provenant de la présentation d'autres objets. Ainsi on associe à un objet une *Ombre Temporelle* pour définir ses aspects temporels, et une "*Ombre Comportementale*" pour ses aspects comportementaux. Celle-ci joue deux rôles principaux: (1) changer le déroulement prédéfini de la présentation en un déroulement suivant différentes alternatives qui correspondent à différents comportements de la présentation et (2) segmenter la présentation en associant des actions à chaque segment.

Notre **modèle de comportements d'une présentation multimédia** propose la modélisation de l'*Ombre Comportementale* qui est définie par un ensemble de comportements. Chaque comportement est modélisé par un type d'événement (déclencheur du comportement), un intervalle de validité et une liste d'actions à exécuter.

Notre modèle de comportements apporte seulement une solution au niveau de la modélisation des comportements, nous avons proposé une implantation objet de ces comportements sous forme d'*objets scénarios* dans un but d'homogénéité de notre approche, mais aussi de réutilisabilité. Cette modélisation objet nous permet une interrogation des comportements à l'aide d'un langage d'interrogation comme OQL. Enfin, nous avons montré comment utiliser la technologie des SGBD actifs pour exécuter nos présentations multimédias actives temporelles. Les règles actives (ou règles Evénement-condition-Action) permettent d'exécuter les différents comportements liés aux objets de la présentation.

Les différents concepts de ces modèles méritaient d'être expérimentés pour montrer leur faisabilité. Cette **expérimentation** s'est faite au-dessus du SGBD à objets existant O_2 et nous avons développé des extensions multimédias. Nous avons implanté nos différentes bibliothèques de classes et développé le gestionnaire de présentations.

Le prototype possède actuellement deux types d'interface qui nous permettent de construire une présentation en créant ou recherchant les objets qui la composent et en les synchronisant. Une fois construite, le gestionnaire de présentations est chargé de la créer et la stocker sous forme d'un objet SO, ensuite il permet de la jouer. Nous pouvons aussi interroger les présentations selon leurs aspects descriptifs, temporels ou

de synchronisation. Tous les exemples de requêtes énoncés au chapitre 5 sont possibles et montrent l'importance d'une base de données pour pouvoir rechercher les données et les présentations multimédias.

6.2 Perspectives

Une utilisation effective des propositions de notre travail passe sûrement d'une part par des améliorations et des expérimentations du prototype développé et d'autre part par des approfondissements d'aspects supplémentaires. Notre domaine de recherche est assez jeune et, à chaque pas effectué, on se rend compte de tout ce qu'il reste à faire pour utiliser au mieux tout ce que peut offrir le multimédia à l'heure actuelle.

Notre première expérimentation a montré la faisabilité des concepts de présentations multimédias que nous avons développé et semble indispensable dans le cadre du multimédia. Mais une extension du prototype doit se faire en particulier par rapport aux points suivants :

- utilisation de notre prototype sur des applications réelles pour montrer que notre approche répond aux besoins des applications et des utilisateurs. Nous avons déjà une première expérience où nous avons utilisé notre prototype pour une application dans le cadre de la conférence BDA'97 qui se déroule à Grenoble. Il s'agit de présenter le programme de chaque journée avec des images, du texte et du son. Elle a montré que notre outil est facilement utilisable pour construire, jouer et rechercher toutes sortes de présentations. Il faut aussi envisager d'adapter notre interface pour différentes applications;

- développer plus la partie sur les comportements pour pouvoir tester leur faisabilité, les règles actives n'ont pas été utilisées, une expérimentation permettrait d'appuyer notre choix pour l'implantation;

- les présentations intentionnelles doivent être intégrées au prototype, leur concept est important si on manipule une base de données avec un grand nombre d'images, de vidéos et de sons. Pour pouvoir présenter un ensemble de ces données, l'utilisation d'une requête est indispensable. De plus, notre gestionnaire de présentations sera capable de l'interpréter dynamiquement pour la présenter à l'utilisateur. On montre ainsi l'importance de l'interrogation des données;

- il est aussi indispensable d'approfondir le type de requêtes que l'on peut utiliser

pour interroger nos présentations. Nous avons indexé nos données multimédias à l'aide de différents mot-clés. Mais les applications requièrent un mécanisme d'indexation plus poussé, il faut aussi intégrer les mécanismes de recherche sur le contenu pour pouvoir interroger les données multimédias sur leur sémantique;

- enfin s'interesser plus à la vérification de la cohérence de nos présentations, nous avons déjà établi un canevas pour cela en réalisant notre atelier de construction de présentations multimédias (Annexe A) qui permet de construire des présentations cohérentes d'un point de vue du modèle.

Les perspectives de recherche sont assez ouvertes dans notre domaine. Dans le cadre de notre projet, un intérêt particulier est porté à la donnée vidéo [Loz96, Loz97, LM97]. Le modèle VStorm s'intéresse à la structure hiérarchique de la vidéo, à l'indexation de son contenu à l'aide d'une base de données à objets qui permet différentes interrogations des vidéos, et enfin à l'édition de nouvelles vidéos appelées "vidéos virtuelles".

Ce travail ouvre plusieurs perspectives. Certains aspects n'ont pas été abordés :

- nous avons surtout développé les aspects temporels de nos présentations, il nous faut intégrer dorénavant des aspects spatiaux sous forme, par exemple, d'une "*Ombre Spatiale*" associée aux objets. La composition spatiale décrit le processus de rassemblement d'objets multimédias sur un écran à un certain moment du temps. De plus, il est aussi possible de définir des contraintes spatiales comme "la présentation des objets `p` et `q` ne doivent pas se chevaucher à l'écran" ou "la présentation de l'objet `p` doit inclure la présentation de l'objet `q`, etc.

- pouvoir décomposer nos présentations multimédias en une structure hiérarchique en se basant sur les recherches pour la donnée vidéo. En effet, il peut être intéressant de décomposer les présentations en différentes scènes et d'offrir un mécanisme d'indexation pour pouvoir les interroger. Certaines scènes provenant de différentes présentations pourraient être retrouvées et réutilisées dans une autre présentation et nous pourrions envisager de les composer entre elles pour construire des "présentations virtuelles" comme pour la donnée vidéo. Ces fonctionnalités permettraient d'améliorer notre *gestionnaire de présentations*;

- un point important à approfondir est l'interrogation des présentations. L'utilisation de la richesse de la sémantique des données multimédias doit permettre une interrogation plus poussée qu'une simple interrogation sur des attributs descriptifs. Nous avons évoqué, au cours de cette thèse, l'utilisation d'expressions de

type GPE (Generalized Path Expressions) qui permettent d'interroger les données sans nécessairement bien connaître le schéma [CACS94]. Cette approche doit être approfondie parce qu'elle permet de dissimuler la structure du schéma et ainsi de pouvoir définir clairement et facilement différents types de requêtes et diminuer leur complexité;

- nous envisageons aussi d'utiliser des données multimédias réparties sur différents sites géographiques, il faut donc étudier les aspects de répartition de données et étudier les problèmes qu'ils posent au niveau de la présentation et au niveau du serveur. Ces données sont souvent hétérogènes ce qui rend complexe leur exploitation. Ces problèmes rejoignent la problématique liée aux entrepôts de données ("data-warehouse"). En effet, la multiplication de sources (hétérogènes) de données que ce soit au sein des entreprises, ou sur le Web pour une large communauté d'usagers, entraîne l'existence de larges entrepôts de données. Les recherches dans ce domaine proposent des solutions pour la constitution, la maintenance et l'exploitation ("data-mining" ou "orpaillage" ou "outils d'investigation") de ces entrepôts, mais elles restent partielles et limitées. Ces recherches peuvent nous être utiles dans la gestion de données multimédias réparties.

Annexe A

L'atelier de construction de présentations multimédias

Cette annexe présente l'outil développé pour la construction de présentations multimédias. Cet outil est construit à partir de trois logiciels existants: GraphTalk, LEdit et le SGBD O_2.

GraphTalk [Par93] est un environnement de développement et d'utilisation d'atelier de génie logiciel graphique, qui permet une programmation graphique des modèles des méthodes. Une programmation GraphTalk produit un Atelier de Génie Logiciel - AGL qui permet la représentation d'un modèle d'un système d'information. L'utilisation de GraphTalk se fait à deux niveaux différents : le niveau méta-modélisation où est créé l'AGL, et le niveau modélisation où l'atelier créé est effectivement utilisé.

La procédure de génération d'un AGL comporte dans GraphTalk quatre parties : (1) la partie *Spécification sémantique* décrit les concepts et les relations entre ces concepts; (2) dans la partie *Affectation des propriétés*, des propriétés peuvent être attachées aux concepts et relations définis dans la spécification sémantique; (3) dans la partie *Spécification des formes*, sont spécifiées les formes que doivent avoir les concepts et les relations déjà définis; (4) la partie *Spécification des fenêtres* permet la spécification d'une interface homme-machine particulière à l'AGL créé.

L'outil LEdit [Par94] est un méta-éditeur qui offre une interface de programmation utilisable lors de la définition d'un éditeur syntaxique pour des langages définis par des grammaires en BNF. L'intégration de GraphTalk et LEdit dans un même environnement rend possible la spécification respectivement des représentations graphiques et des représentations textuelles de l'outil qui sera créé.

Dans l'état actuel de développement du prototype les outils GraphTalk/LEdit et O_2 tournent sur des machines différentes et communiquent à travers une plate-forme de communication intégrée au prototype, et qui est construite à l'aide de sockets. Ainsi lorsque GraphTalk a besoin d'une information qui est stockée dans la base O_2, il la demande via les sockets qui seront egalement utilisées par O_2 pour envoyer la réponse. Chaque logiciel utilise une couche intermédiaire (GraphTalk : API et O_2 : STORM Engine) entre lui et la plate-forme de communication; dans ces couches sont définis les "démons" déclarés au travers de programmes C/C++. L'architecture de l'outil est présentée sur la figure A.1.

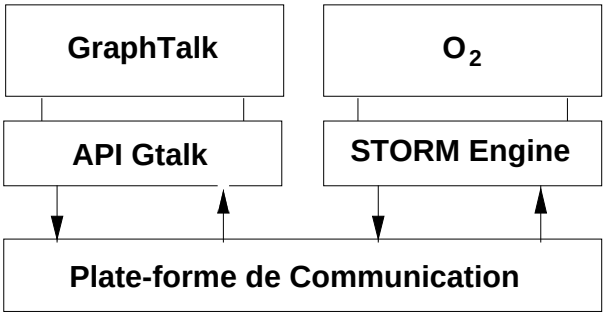

FIG. A.1 – *Architecture de l'Outil*

À travers l'API ("Application Programming Interface") de GraphTalk les fonctionnalités de cet outil peuvent être étendues. Ainsi on peut ajouter des démons qui sont déclenchés automatiquement par des événements donnés (après création d'un noeud, avant suppression, etc.). Ce sont ces démons qui nous utilisons pour implanter la communication avec O_2. Le STORM engine correspond au gestionnaire de présentations que nous avons développé au dessus du SGBD O_2 (et que nous décrivons au cours du chapitre 5).

Par rapport à la division de taches lors d'une modélisation, GraphTalk/LEdit s'occupe du processus de modélisation et de l'interface homme-machine de l'atelier. Le processus de modélisation est exécuté d'une manière totalement graphique avec les icones présentés sur la figure A.2.

Ces composants graphiques peuvent être divisés en cinq types différents : (1) la *Présentation* (premier icone à gauche de la première ligne de la figure A.2) qui représente la racine d'une présentation; (2) les *Opérateurs* (les sept autres icones de la première ligne de la figure A.2) représentent les opérateurs séquentiels et parallèles. Les cinq icones à droite de la première ligne représentent les types d'opérateurs parallèles *par-*

overlap, par-during, par-finish, par-start et *par-equal*; le deuxième et le troisième icones de cette ligne représentent les opérateurs séquentiels *seq-meet* et *seq-before*; (3) les *Objets Monomédias* (les quatre icones à gauche de la deuxième ligne) représentent respectivement des instances des classes `MyImage`, `MyText`, `MyAudio` et `MyVideo`; (4) le *Show* (le premier icone à droite de la deuxième ligne) représente une instance de la classe `Show`; (5) le *SO_Query* (l'icone à droite de la deuxième ligne) est utilisé pour représenter les requêtes modélisées dans l'outil. Une telle requête est composée d'une représentation graphique qui donne son comportement temporelle, associée à un composant textuel qui montre la requête en OQL (les objets de la requête). La figure A.3 présente les éléments utilisés pour modéliser une requête.

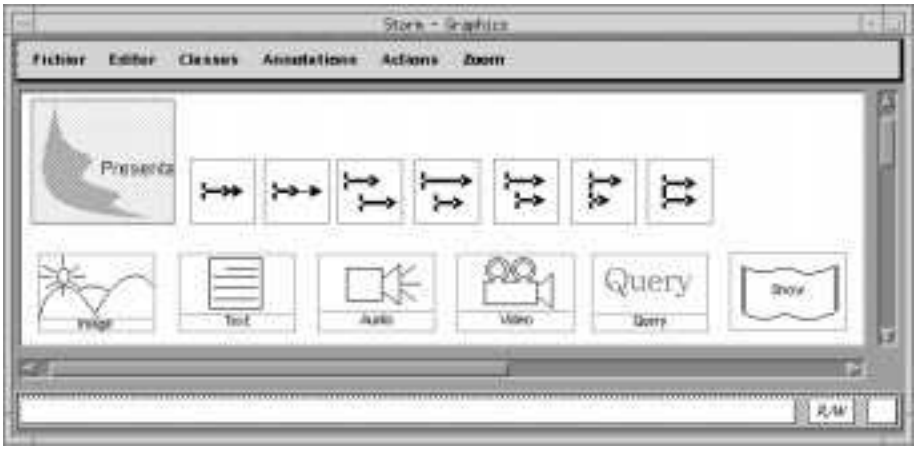

FIG. A.2 – *Composants Graphiques de l'Outil*

L'icone `SO_Query` dans l'outil est donné par un graphe à part : le fait de cliquer sur un objet de ce type dans l'atelier produit deux effets : (1) l'ouverture d'une fenêtre ("SOQuery - Query" de la figure A.3) pour la représentation graphique de la structure temporelle de la requête et (2) l'ouverture d'une autre fenêtre ("sql - <untitled>" de la figure A.3) pour la déclaration en OQL de la requête; cette partie déclarative est implantée avec l'outil LEdit.

La première ligne de la fenêtre "SOQuery - Query" de la figure A.3 montre les opérateurs qui peuvent être utilisés dans une requête; ils sont différents de ceux de la figure A.2 car, de type ensembliste, ils permettent des opérations sur des collections (les opérateurs introduits à la figure A.2 ne sont pas présentés mais ils peuvent être utilisés dans une requête). Les autres éléments de cette figure sont utilisés pour représenter des objets référencés dans une requête (ils ne sont pas des objets monomédias stockés dans la base O_2). L'élément `Q_Type` est utilisé pour représenter des types de base (ex:

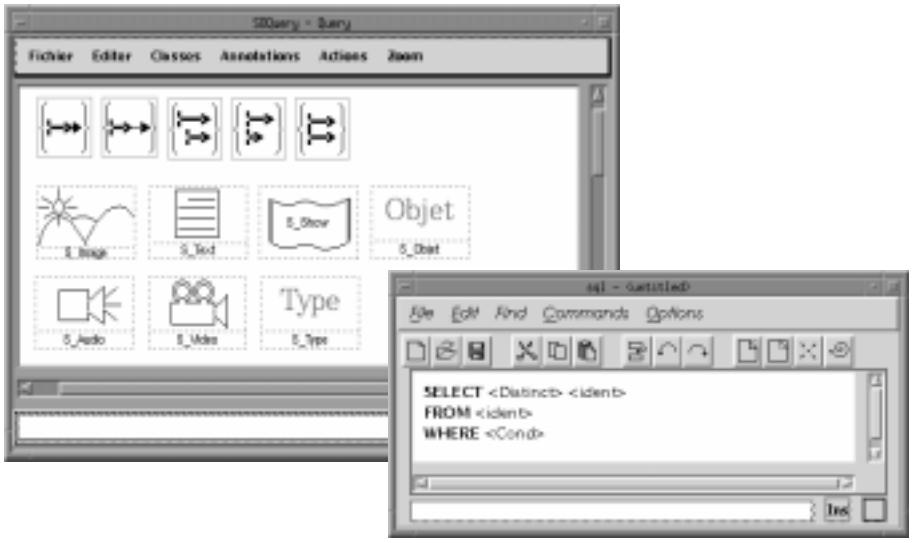

FIG. A.3 – *Composants d'une Requête*

réel, entier, etc.). Le dernier élément, le `Q_Objet` pour représenter des classes qui ne sont pas des classes SO.

La création d'une présentation débute par la création de l'icone `Présentation`. Ensuite il faut lui attacher un opérateur (séquentiel ou parallèle) et à celui-ci d'autres opérateurs, ou des objets monomédias ou des requêtes.

Lorsque nous voulons extraire de la base de données un objet monomédia, GraphTalk via son API et la plate-forme de communication demandeau STORM Engine une liste avec tous les objets de la base qui correspondent au type d'objet monomédia demandé. Le STORM Engine via la plate-forme de communication renvoie la liste demandée. La création par exemple d'un noeud dans GraphTalk pour représenter une donnée monomédia de type vidéo, déclenche un appel via la plate-forme de communication au STORM Engine. Celui-ci envoie une requête à O_2; le SGBD répond à cette requête en renvoyant comme résultat toutes les références à des données monomédias de type vidéo qui sont stockés dans la base. L'objet choisi est alors affiché dans l'atelier et une "image" de celui-ci est stockée dans la base. Le processus de modélisation d'une requête est indépendant de la base, car les objets déclarés dans une requête ne sont pas des objets monomédias de la base mais de références. Après avoir écrit la requête, on doit l'envoyer à O_2 pour que celui-ci puisse l'évaluer et renvoyer le type du résultat (ex: $list(Image)$); cette réponse est affichée graphiquement en respectant l'interprétation par défaut des constructeurs. Les composants créés (une image, un show, un objet,

etc.) ne peuvent pas être effacés, mais on peut changer des opérateurs et redéfinir un comportement temporel différent du défaut.

Après la construction d'une présentation dans l'atelier, GraphTalk envoie un script qui décrit celle-ci au STORM Engine pour que celui-ci puisse créer et jouer la présentation. Ce script est en fait une instance de la classe `Show` qui décrit quelle est la synchronisation entre les différents objets monomédias de la modélisation. Si la modélisation comporte un icone `SO_Query` pour représenter une requête, l'outil introduit un sous-arbre qui représente cette requête dans le script. Le script est envoyé via la plate-forme de communication et le STORM Engine après l'avoir reçu et traité, envoie une requête au SGBD O_2 pour récupérer les composants de la présentation; ensuite le STORM Engine en respectant la synchronisation établie dans le script joue la présentation modélisée.

Nous pouvons dans l'atelier "jouer" n'importe quel objet qui fait partie de la présentation à travers un processus semblable à celui de le définition des objets. GraphTalk envoie une demande au STORM Engine qui va jouer l'objet demandé.

L'exemple présenté sur la figure A.4 modélise une présentation composée de quatre objets monomédias: l'audio `My Way`, l'image `I_Meribel`, la vidéo `V_Meribel` et le texte `T_Meribel`. L'opérateur `Seq1` définit une présentation séquentielle entre le son `MyWay` et le reste de la présentation dont la racine est `Par1`. Les opérateurs `Par1` et `Par2` décrivent une synchronisation parallèle entre `V_Meribel`, `I_Meribel` et `T_Meribel`. Enfin, sur la figure apparaissent deux menus, dont l'un permet la création de la présentation avec l'option "CreerPresentation" et l'autre de jouer un objet de la présentation avec l'option "jouer_Text" (dans ce cas-là, il s'agit de jouer le texte `T_Meribel`).

En conclusion, notre environnement permet de construire des applications multimédias combinant et synchronisant différents types d'objets. La principale contribution de ce travail se situe dans le lien qui a été établi entre un atelier de modélisation et un système de gestion de base de données dans le cadre d'applications multimédias.

L'atelier de conception est utilisé pour spécifier la construction de la présentation d'un point de vue conceptuel. On construit un arbre où les feuilles sont les objets à présenter et les noeuds les contraintes de synchronisation. Pour les objets requêtes, l'atelier est essentiel car la structure temporelle de leur présentation ne peut être faite en référençant directement les objets. Nous construisons donc un arbre qui spécifie le comportement temporel et qui permet de contrôler celui-ci en fonction du type.

L'utilisation d'un atelier de conception pour la construction de telles applications permet d'assurer que celle-ci respecte les règles de cohérence vis à vis du modèle de

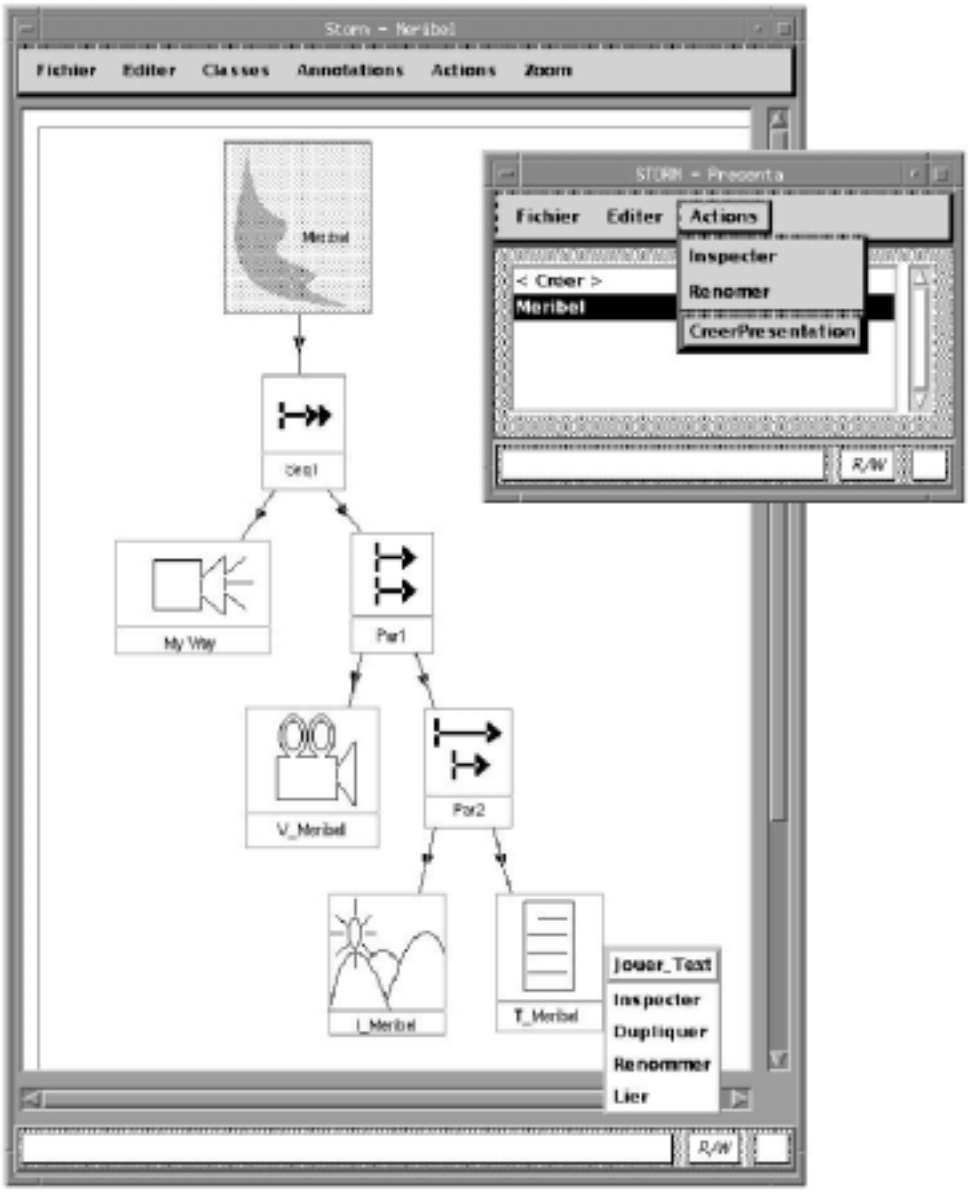

FIG. A.4 – *Exemple de Modélisation*

présentations. De plus, la visualisation graphique de la présentation sépare les aspects instances (sélection des objets participant à la présentation) des aspects temporels. Ceci favorise une démarche incrémentale pour la conception et permet des retours arrières en gardant une vue globale de la présentation. Cela permet également la création de modèles standards de présentation qui peuvent être instanciés selon les besoins.

Annexe B

Les notations en OMT

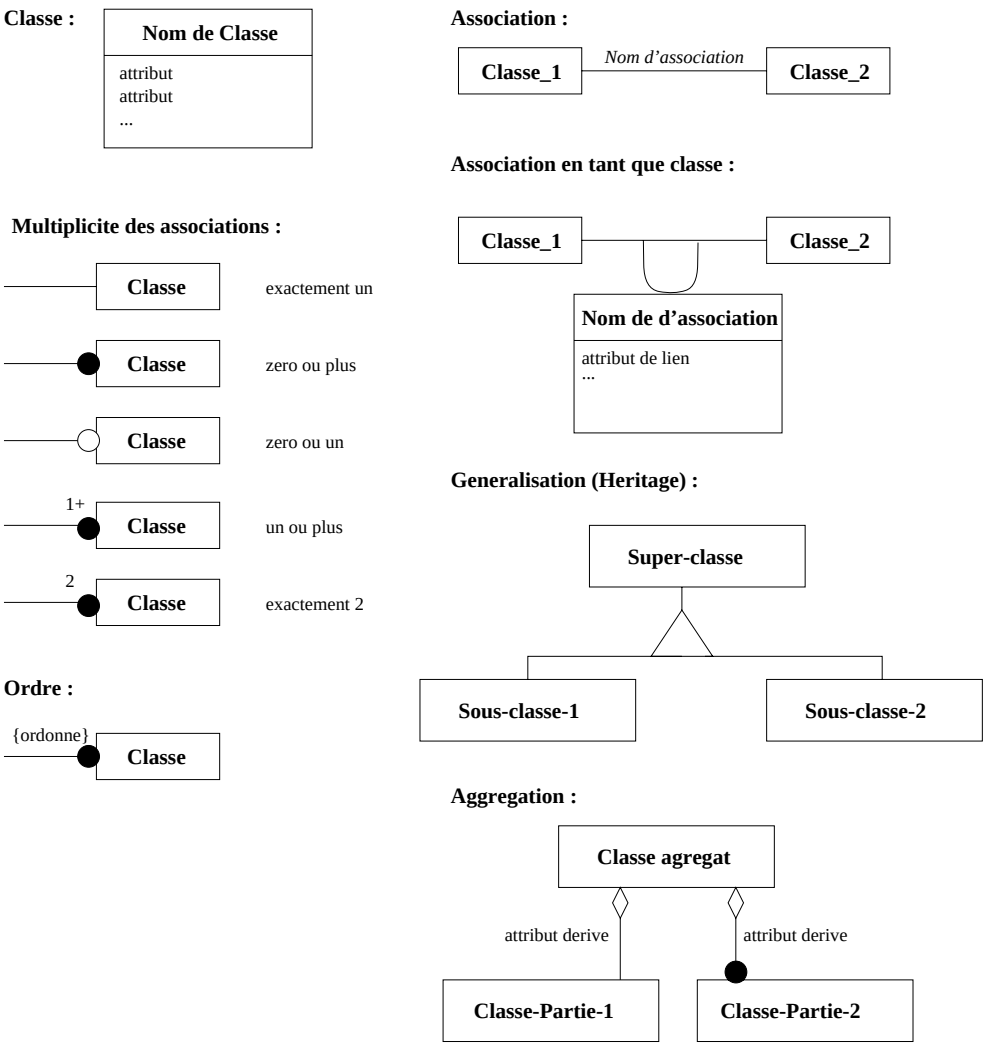

FIG. B.1 – *Modèle objet*

Bibliographie

[ABF+95] J. Ashley, R. Barber, M. Flickner, J. Hafner et D. Lee. – Automatic and Semi-Automatic Methods for Image Annotation and Retrieval in QBIC. – Dans *SPIE'95, Image and Video Storage Retrieval III*, 1995.

[ABH97] P.M.G. Apers, H.M. Blanken et M.A.W. Houtsma, éditeurs. – *Multimedia Databases in Perspective*. – Springer, 1997.

[AC93] M. Adiba et C. Collet. – *Objets et Bases de Données: le SGBD O_2*. – Hermès, 1993.

[AC95] A. Analyti et S. Christodoulakis. – Multimedia object modelling and content-based querying. – Dans *Advanced Course Multimedia Databases in Perspective*, The netherlands, juin 1995.

[ACC+96] S. Adali, K.S. Candan, S. Chen, K. Erol et V.S. Subrahmanian. – The advanced video information system : data structures and query processing. – *Multimedia systems*, pages 172–186, août 1996.

[Adi90] M. Adiba. – Management of multimedia complex objects in the '90s. – *Lecture Notes in Computer Science N466, Database systems of the 90s*, 1990.

[Adi95] M. Adiba. – STORM: Structural and Temporal Object-oRiented Multimedia database system. – Dans *IEEE International Workshop on Multimedia DBMS, Minnowbrook Conference Center*, pages 10–15, Blue Mountain Lake, NY, USA, août 1995 (version étendue au congrès bases de données avancées à Nancy en août 1995).

[Adi96] M. Adiba. – *STORM: an Object-Oriented Multimedia DBMS*, chapitre 3 du livre Multimedia Database Systems: design and implementation

strategies. – K. Nwosu, B. Thuraisingham et B. Berra, Kluwer Academic Publishers édition, mai 1996.

[AH91] D.P. Anderson et G. Homsy. – A continuous media I/O server and its synchronization mechanism. – *Computer*, Octobre 1991.

[AK94] K. Aberer et W. Klas. – Supporting Temporal multimedia Operations in Object-Oriented Database System. – Dans *International Conference on Multimedia Computing and Systems*, Boston, mai 1994.

[AL95] D. A. Adjeroh et M.C. Lee. – Synchronization Mechanisms for Distributed Multimedia Presentation Systems. – Dans *IEEE International Workshop on Multimedia DBMS, Minnowbrook Conference Center*, Blue Mountain Lake, NY, USA, août 1995.

[AL96] D. A. Adjeroh et M.C. Lee. – *Synchronization and user interaction in distributed multimedia presentation systems*, chapitre du livre Multimedia Database Systems: design and implementation strategies. – K. Nwosu, B. Thuraisingham et B. Berra, Kluwer Academic Publishers édition, mai 1996.

[All83] J.F. Allen. – Maintaining knowledge about temporal intervals. – *CACM*, 26(11), 1983.

[ALMM97] M. Adiba, R. Lozano, H. Martin et F. Mocellin. – Management of multimedia data using an object-oriented database system. – Dans *DEXA QPMIDS workshop 1997*, Toulouse, septembre 1997.

[AM97] M. Adiba et F. Mocellin. – STORM : une approche à objets pour les Bases de Données Multimédias. – *Technique et Science Informatiques*, 16(7), 1997.

[AN86] M. Adiba et B. Q. Ngoc. – Historical multimedia databases. – Dans *VLDB*, Kyoto (Japon), 1986.

[BCF95] E. Bertino, B. Catania et E. Ferrari. – Research Issues in Multimedia Query Processing. – Dans *Advanced Course Multimedia Databases in Perspective*, The netherlands, juin 1995.

[BDK92] F. Bancilhon, C. Delobel et P. Kanellakis. – *Building an Object-Oriented Database System: The story of O_2*. – Morgan Kaufmann, 1992.

[BGMLS93] P. B. Berra, F. Golshani, R. Mehrotra et O.R. Liu Sheng. – Introduction to Multimedia Information Systems. – Dans *IEEE Transactions on Knowledge and Data Engineering*, volume 5, août 1993.

[BT93] P. Boursier et P.-A. Taufour. – *La Technologie Multimédia.* – Hermès, 1993.

[BZ93] M.C. Buchanan et P. Zellweger. – Automatic temporal layout mechanisms. – Dans *First International Conference on Multimedia*, Anaheim, California, août 1993.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet et M. Scholl. – From structured documents to novel query facilities. – *SIGMOD*, 1994.

[Cat93] R. Cattel, éditeur. – *Object Databases: the ODMG93 Standard.* – Morgan-Kaufmann, 1993.

[CC96a] S. T. Campbell et S. M. Chung. – *Database approach for the management of multimedia information*, chapitre du livre Multimedia Database Systems: design and implementation strategies. – K. Nwosu, B. Thuraisingham et B. Berra, Kluwer Academic Publishers édition, mai 1996.

[CC96b] C. Collet et T. Coupaye. – Composite Events in NAOS. – Dans *Proc. of the 7th Int. Conf. on Database and Expert Systems Applications(DEXA'96)*, Zurich - Switzerland, septembre 1996.

[CC96c] C. Collet et T. Coupaye. – Primitive and Composite Events in NAOS. – Dans *les Actes des 12èmes Journées Bases de Données Avancées*, Cassis, août 1996.

[CC96d] R. Cutler et K. S. Candan. – *Multimedia Authoring Systems*, chapitre du livre Multimedia Database Systems, pages 279–296. – Subrahmanian, V.S. and Jajodia, S., Springer édition, 1996.

[CC97] H-J Chang et S-K Chang. – *Temporal Modeling and Inter-Media Synchronization for Presentation of Multimedia Systems*, chapitre 15 du livre Multimedia Information Storage and Management, pages 373–398. – Kluwer Academic Publishers, 1997.

[CCS94] C. Collet, T. Coupaye et T. Svensen. – NAOS: Efficient and modular reactive capabilities in an object-oriented database system. – Dans *VLDB Conference*, Chili, septembre 1994.

[CHCA94] C. Collet, P. Habraken, T. Coupaye et M. Adiba. – Active rules for the software engineering platform goodstep. – Dans *2nd International Workshop on Database and Software Engineering, 16th Int. Conf. on Software Engineering*, Sorrento, Italy, mai 1994.

[CHN+95] W.F. Cody, L.M. Haas, W. Niblack, M. Arya, M.J. Carey, F. Ragin, M. Flickner et co. – Querying Multimedia Data from Multiple Repositories by Content: the Garlic Project. – Dans *IFIP 2.6 Third Working Conference on Visual Database Systems (VDB-3)*, Lausanne, mars 1995.

[Chr94] S. Christodoulakis. – Multimedia Information Systems. – Dans *EDBT Conference*, Cambridge, mars 1994. – Tutorial notes.

[CHR96] C. Collet, P. Habraken et C. Roncancio. – Règles actives dans les SGBD. – *Ingénierie des Systèmes d'Information*, 4(3), 1996.

[Chu96] S. M. Chung, éditeur. – *Multimedia Information Storage and Management.* – Kluwer Academic Publishers, 1996.

[CKAK94] S. Chakravarthy, V. Krishnaprasad, E. Anwar et S-K. Kim. – Composite Events for Active Databases: Semantics, Contexts and Detection. – Dans *Proc. of the 20th Int. Conf. on Very Large Data Bases*, pages 606–617, Santiago, Chili, Septembre 1994.

[CL92] S. Chakravarthy et D. Lomet. – Special issue on active databases. – Dans *Proc. IEEE Data Engineering Bulletin*, volume 15, 1992.

[CM93] S. Chakravarthy et D. Mishra. – Snoop : An Expressive Event Specification Language For Active Databases. – Rapport de recherche, University of Florida, Gainesville, USA, March 1993.

[CMCSCL95] C.Y.R. Chen, D.S. Meliksetian, M. Cheng-Sheng Chang et L.J. Liu. – Design of a multimedia object-oriented DBMS. – *Multimedia Systems*, 3(5/6):217–227, Novembre 1995.

[Col96] C. Collet. – *Bases de Données Actives : des systèmes relationnels aux systèmes à objets.* – Diplome D'Habilitation à Diriger les Recherches, LSR-IMAG, Université Joseph Fourier, octobre 1996.

[Col97] C. Collet. – Active rules for databases. – *Springer Verlag*, octobre 1997.

[Com91] Apple Computer. – Quicktime Developper's Guide. – Rapport de recherche, 1991.

[Cou96] T. Coupaye. – *Un modèle d'exécution paramétrique pour systèmes de bases de données actifs.* – Thèse, Université Joseph Fourier, Grenoble, novembre 1996.

[DDB91] K. Dittrich, U. Dayal et A.P. Buchman. – An object-oriented database systems. – Dans *Topics in Information Systems*, Springer Verlag, 1991.

[DHB96] C. Djeraba, K. Hadouda et H. Briand. – Management of multimedia scenario in an object-oriented database. – Dans *Proc. IEEE International Workshop on Multi-Media Data Base Management Systems*, Blue Mountain Lake, NewYork, août 1996.

[DK95] A. Duda et C. Keramane. – Structured Temporal Composition of Multimedia Data. – Dans *Proceedings of the 1st IEEE International Workshop for MM-DBMSs*, Blue Mountain Lake, août 1995.

[Fay97] L. Fayolle. – *Définition et détection d'événements dans NAOS.* – Mémoire Ingénieur CNAM, Conservatoire National des Arts et Métiers, Grenoble, juin 1997.

[FLM97] J-C Freire, R. Lozano et F. Mocellin. – Vers un Atelier de Structuration et Construction de Présentations Multimédias. – Dans *Congrès Inforsid'97*, Toulouse, juin 1997.

[FLMM97] J-C Freire, R. Lozano, H. Martin et F. Mocellin. – A STORM Environment for Building Multimedia Presentations. – Dans *the 12th International Conference on Information Networking (ICOIN-12)*, Koganei, Tokyo, Japon, janvier 1997.

[FSNA95] M. Flickner, H. Sawhney, W. Niblack et J. Ashley. – Query by image and video content: The QBIC system. – *Computer*, septembre 1995.

[GBT92] S. Gibbs, C. Breiteneder et D. Tsichritzis. – Modelling of audio/video data. – Dans *11th Conf. on the Entity Relationship approach*, Karlsruhe, octobre 1992.

[GBT93] C. Gibbs, C. Breiteneder et D. Tsichritzis. – Audio/Video Databases : An Object-Oriented Approach. – Dans *Proc. of IEEE Ninth International Conference on Data engineering*, Vienna, avril 1993.

[GBT94] S. Gibbs, C. Breiteneder et D. Tsichritzis. – Data modelling of time-based media. – *SIGMOD*, 1994.

[GDT94] S. Gibbs, C. Dionysios et D. Tsichritzis. – *Multimedia programming: objects, environments and frameworks.* – Addison-Wesley, 1994.

[Gha94] A. Ghafoor. – Multimedia Database Systems. – *Tutorial notes, Data Engineering Conference*, 1994.

[Gha95] A. Ghafoor. – Multimedia Database Management Systems. – *ACM Computing Surveys*, 4(27):593–598, décembre 1995.

[GMY93] J. Griffioen, R. Mehrotra et R. Yavatkar. – An object-oriented model for image information representation. – *CIKM*, 1993.

[GR94] S.M. Gandhi et E.L. Robertson. – A Data Model for Audio-Video Data. – Dans *COMAD*, Bangalore, India, décembre 1994.

[GRG95] S.M. Gandhi, E.L. Robertson et D.V. Gucht. – Modeling and querying primitives for digital media. – Dans *IEEE International Workshop on Multimedia DBMS*, Minnowbrook Conference Center, Blue Mountain Lake, NY, USA, août 1995.

[HFK95] N. Hirzalla, B. Falchuk et A. Karmouch. – A Temporal Model for Interactive Multimedia Scenarios. – Dans *IEEE.* , 1995.

[HK96] N. Hirzalla et A. Karmouch. – *A Multimedia Query Specification Language*, chapitre du livre Multimedia Database Systems: design and implementation strategies. – K. Nwosu, B. Thuraisingham et B. Berra, Kluwer Academic Publishers édition, mai 1996.

[HR96] S. Hibino et A. Rundensteiner. – *A visual multimedia query language for temporal analysis of video data*, chapitre 5 du livre Multimedia Database Systems: design and implementation strategies. – K. Nwosu, B. Thuraisingham et B. Berra, Kluwer Academic Publishers édition, mai 1996.

[HSR94] R. Hamakawa, H. Sakagami et J. Rekimoto. – Mbuild : Multimedia data builder with box and glue. – Dans *International Conference on Multimedia Computing and Systems*, Boston, Mass., mai 1994.

[IDG94] M. Iino, Y.F. Day et A. Ghafoor. – An object-oriented model for spatio-temporal synchronization of multimedia information. – Dans *International Conference on Multimedia Computing and Systems*, Boston, Mass., mai 1994.

[ISO92] ISO. – Iso/iec 10744 is. – Rapport de recherche, août 1992.

[Jag96] H.V. Jagadish. – *Indexing for Retrieval by Similarity*, chapitre du livre Multimedia Database Systems, Issues and Research Directions, pages 166–184. – Springer-Verlag, V.S. Subrahmanian et S. Jajodia édition, 1996.

[JLSI97] M. Jourdan, N. Layaïda et L. Sabry-Ismail. – Time Representation and Management in MADEUS: an authoring environment for multimedia documents. – Dans *Multimedia Computing and Networking*. SPIE, février 1997.

[KA95] W. Klas et K. Aberer. – Multimedia Applications and their Implications on Database Architectures. – Dans *Advanced Course Multimedia Databases in Perspective*, The netherlands, juin 1995.

[KB96] S. Khoshafian et B. Baker. – *Multimedia and Imaging Databases*. – Morgan Kaufmann Publishers, 1996.

[KD96] C. Keramane et A. Duda. – Interval Expressions, a functional model for interactive dynamic multimedia presentations. – Dans *Proc. IEEE Int. Conference on Multimedia Computing and Systems*, Hiroshima, juin 1996.

[KEM97] A. Karmouch, J. Emery et O. Megzari. – *Architecture and Storage Model for Multimedia Documents*, chapitre 14 du livre Multimedia Information Storage and Management, pages 346–371. – Kluwer Academic Publishers, 1997.

[KL96] T. Kwon et S. Lee. – *Load-Balanced Data Placement for Variable-Rate Continuous Media Retrieval*, chapitre 7 du livre Multimedia Database Systems: design and implementation strategies. – K. Nwosu, B. Thuraisingham et B. Berra, Kluwer Academic Publishers édition, may 1996.

[KRRK93] J.F. Koegel, L.W. Rutledge, J.L. Rutledge et C. Keskin. – HyOctane: a HyTime engine for an MMIS. – Dans *First ACM International Conference on Multimedia*, Anaheim, Ca., août 1993.

[KT94] S.C. Kau et J. Tseng. – MQL - A Query Language for Multimedia Databases. – Dans *Second ACM International Conference on Multimedia*, pages 511–516. ACM Press, 1994.

[LAFG93] T.D.C. Little, G. Ahanger, R.J. Folz et J.F. Gibbon. – A digital on-demand video service supporting content-based queries. – Dans *First ACM International Conference on Multimedia*, Anaheim, Ca., août 1993.

[Lay96] N. Layaïda. – Issues in temporal representations of multimedia documents. – Dans *Workshop on Real Time Multimedia and the Web 96*, INRIA Sophia-Antipolis, octobre 1996.

[LG90] T.D. Little et A. Ghafoor. – Synchronisation and Storage Models for Multimedia Objects. – *Journal on Selected Areas in Communication*, 8(3), 1990.

[LG91a] D. Le Gall. – MPEG: A video compression standard for multimedia applications. – *CACM*, avril 1991.

[LG91b] T.D. Little et A. Ghafoor. – Spatio-Temporal composition of distributed multimedia objects for value-added networks. – *Computer*, octobre 1991.

[LG93] T.D. Little et A. Ghafoor. – Interval-based conceptual models for time-dependent multimedia data. – Dans *IEEE Transactions on knowledge and data engineering*, volume 5. août 1993.

[LK95] N. Layaïda et C. Keramane. – Maintaining Temporal Consistency of Multimedia Documents. – Dans *ACM Workshop on Effective Abstractions in Multimedia*, San Franscico, California, novembre 1995.

[LKG94] L. Li, A. Karmouch et N.D. Georganas. – Multimedia Teleorchestra with Independent Sources: Part 1 - Temporal Modeling of Collaborative Multimedia Scenarios. – *Multimedia Systems*, pages 143–153, 1994.

[LM97] R. Lozano et H. Martin. – Querying Virtual Videos Using Path and Temporal Expressions. – Dans *SAC '98, ACM Symposium on Applied Computing*, Marriott Marquis, Atlanta, Georgia, U. S. A., février 1997.

[Loz96] R. Lozano. – *Système de Gestion de Bases de Données Multimédias et Vidéos.* – Rapport de DEA, Grenoble, juin 1996.

[Loz97] R. Lozano. – Vstorm, un modèle pour la donnée vidéo. – Dans *Journée O_2 de la conférence BDA'97*, Grenoble, septembre 1997.

[LR94] M. Löhr et T.C. Rakow. – Audio support for an object-oriented database management system. – Rapport de recherche, GMD, St. Augustin, décembre 1994.

[LSI96a] N. Layaïda et L. Sabry-Ismail. – MADEUS : un modèle de documents multimédia structurés. – à paraitre dans Techniques et Sciences Informatiques (TSI), numéro thématique Multimédia Collecticiels, 1996.

[LSI96b] N. Layaïda et L. Sabry-Ismail. – Maintaining temporal consistency of multimedia documents using constraint networks. – Dans *Multimedia Computing and Networking*, octobre 1996.

[Mar96] S. Marcus. – *Multimedia Database Systems, Issues and Research Directions*, chapitre du livre Querying Multimedia Databases in SQL, pages 263–277. – Springer-Verlag, V.S. Subrahmanian et S. Jajodia édition, 1996.

[Mar97] H. Martin. – Specification of intentional presentations using an object-oriented database. – Dans *DMIB'97, International Symposium on Digital Media Information Base*, Nara, Japon, novembre 1997.

[MBE95] T. Meyer-Boudnik et W. Effelsberg. – MHEG explained. – Dans *IEEE*, 1995.

[MM94] S.R.L. Meira et A.E.L. Moura. – A scripting language for multimedia presentation. – Dans *International Conference on Multimedia Computing and Systems*, Boston, Mass., mai 1994.

[MMA96] F. Mocellin, H. Martin et M. Adiba. – STORM : a Structural and Temporal Object-oRiented Multimedia database system. – Dans *Conference EDBT96, demonstration and poster*, Avignon, mars 1996.

[Moc96a] F. Mocellin. – Extensions multimédias au SGBD O_2. – Dans *Journée O_2 dans le cadre du Congrès INFORSID'96*, Bordeaux, juin 1996.

[Moc96b] F. Mocellin. – Présentations multimédias dans un SGBD Orienté-Objet. – Dans *Journée Jeunes Chercheurs du GDR-BD*, Paris, novembre 1996.

[NTB96] K. Nwosu, B. Thuraisingham et B. Berra. – *Multimedia Database Systems: design and implementation strategies.* – Kluwer Academic Publishers édition, mai 1996.

[NXN91] S.R. Newcomb, N.A. Xipp et V.T. Newcomb. – The Hytime Hypermedia Time-based Document structuring language. – *CACM*, 34(11), novembre 1991.

[ORA96] B. Ozden, R. Rastogi et Silberschatz A. – *The Storage and Retrieval of Continuous Media Data*, chapitre du livre Multimedia Database Systems, Issues and Research Directions, pages 237–261. – Springer-Verlag, V.S. Subrahmanian and S. Jajodia édition, 1996.

[OS95] V. E. Ogle et M. Stonebraker. – Chabot: Retrieval from a relational database of images. – *Computer*, septembre 1995.

[OSEMV95] M.T. Ozsu, D. Szafron, G. El-Medani et C. Vittal. – An object-oriented multimedia database system for a news-on-demand application. – *Multimedia Systems*, 3(5/6):182–203, novembre 1995.

[OT93] E. Oomoto et K. Tanaka. – OVID: design and implementation of a video-object database system. – Dans *IEEE Transactions on Knowledge and Data Engineering.* août 1993.

[Par93] Software Technologies Parallax. – Graphtalk 2.5 - interface de programmation, 1993.

[Par94] Software Technologies Parallax. – Ledit - interface de programmation, 1994.

[PR93] B. Prabhakaran et S.V. Raghavan. – Synchronization models for multimedia presentations with user participation. – Dans *First ACM International Conference on Multimedia*, Anaheim, Ca., août 1993.

[Pri93] R. Price. – MHEG: an introduction to the future international standard for hypermedia object interchange. – Dans *First ACM International Conference on Multimedia*, Anaheim, Ca., août 1993.

[Ran93] P. Rangan. – Efficient storage techniques for digital continuous multimedia. – *IEEE Transactions on Knowledge and Data Engineering*, 1993.

[RNL95] T.C. Rakow, E.J. Neuhold et M. Loehr. – Multimedia Database Systems - The Notions and the Issues. – Dans *Datenbanksysteme in Buro, Technik and Wissenschaft (BTW)*, pages 1–29, Dresden, Germany, mars 1995. Springer.

[Rum95] J. Rumbaugh. – *OMT Modélisation et Conception Orientées Objet.* – Masson, 1995.

[Sa94a] K. Süllow et al. – Multimedia forum : an Interactive Online Journal. – Dans *Proc. of the international conference on electronik publishing, Document manipultaion, and Typography, EP94*, Darmstadt, Germany, 1994.

[Sa94b] J.-J. Sung et al. – Hypermedia Information Retrieval System Using MHEG Coded Environment. – Dans *Lecture Notes in Computer Science*, volume 868, pages 67–77, Berlin, 1994.

[SLR94] P. Seshadri, M. Livny et R. Ramakrishnan. – Sequence query processing. – *SIGMOD*, 1994.

[Tec93] O_2 Technology. – *The O_2 User Manual (Version 4.2.).* – Versailles, France, 1993.

[TRR94] H. Thimm, K. Röhr et T.C. Rakow. – A Mail-based Teleservice Architecture for Archiving and Retrieving Dynamically Composable Multimedia Documents. – Dans *Proceedings of the conference on multimedia transport and teleservices, MMTT94*, 1994.

[Vaz96] M. Vazirgiannis. – *An object-oriented modeling of multimedia database objects and applications*, chapitre du livre Multimedia Database Systems: design and implementation strategies. – K. Nwosu, B. Thuraisingham et B. Berra, Kluwer Academic Publishers édition, mai 1996.

[VB95] P.A.C. Verkoulen et H.M. Blanken. – SGML/HyTime for supporting cooperative authoring of multimedia applications. – Dans *Advanced Course Multimedia Databases in Perspective*, The netherlands, juin 1995.

[VH93] M. Vazirgiannis et M. Hatzopoulos. – A Script Based Approach for Interactive Multimedia Applications. – Dans *International Conference on Multi-Media Modelling*, Singapore, novembre 1993.

[VH95] M. Vazirgiannis et M. Hatzopoulos. – Integrated Multimedia Object and Application Modelling Based on Events and Scenarios. – Dans *IEEE International Workshop on Multimedia DBMS, Minnowbrook Conference Center*, pages 48–55, Blue Mountain Lake, NY, USA, août 1995.

[vRa93] G. van Rossum et al. – Cmifed: A Presentation Environment for Portable Hypermedia Documents. – Dans *Proc. of the first ACM Int. Conference on Multimedia*, pages 183–188, 1993.

[VS96] M. Vazirgiannis et T. Sellis. – Event and Action Representation and Composition for Multimedia Application Scenario Modelling. – Dans E. Moeller et H. Push B. Butscher, éditeur, *Lecture notes in computer science*, volume 1045, pages 71–89. mars 1996.

[WDG94] R. Weiss, A. Duda et D.K. Gifford. – Content-based access to algebraic video. – Dans *International Conference on Multimedia Computing and Systems*, Boston, Mass., mai 1994.

[WDG95] R. Weiss, A. Duda et D.K. Gifford. – Composition and Search with a Video Algebra. – Dans Spring, éditeur, *IEEE multimedia*, pages 12–25. 1995.

[WK87] D. Woelk et W. Kim. – Multimedia information management in an object-oriented database system. – Dans *Proc. of the 13th International Conference on Very Large Databases*, 1987.

[WR94] T. Wahl et K. Rothermel. – Representing time in multimedia systems. – Dans *International Conference on Multimedia Computing and Systems*, Boston, Mass., mai 1994.