N° d'ordre : 2642

Thèse

présentée

devant l'Université de Rennes I

pour obtenir

le grade de : **Docteur de l'Université de Rennes I**

Mention : *Informatique*

par

Olivier Aubert

Équipe d'accueil :	Laboratoire d'Informatique des Télécommunications, ENST de Bretagne, Brest
École doctorale :	Mathématiques, Informatique, Signal et Électronique et Télécommunications
Composante universitaire :	IFSIC

TITRE DE LA THÈSE :

Patron de conception pour l'analyse et la construction de systèmes à comportements autoadaptatifs

---

soutenue le 21 décembre 2001 devant la commission d'examen

Composition du jury :

Rapporteurs :	M. Jean Bézivin
	M. Mesaac Makpangou
Examinateurs :	Mme Françoise André
	M. Antoine Beugnard
	M. Michel Raynal

Table des matières

Table des figures

Introduction

La popularité croissante du WWW a mis à l'épreuve les capacités de l'infrastructure le supportant, tant matérielles que logicielles. Sa diffusion à l'échelle mondiale pose des enjeux de passage à l'échelle (*scalability*), tant dans le transport des informations que dans leur traitement ou leur localisation. Jusqu'ici, les nouveaux besoins en termes de diffusion ont pu être couverts par l'augmentation des capacités de transport ou de traitement, ou par l'introduction de nouvelles techniques telles que des systèmes de caches. Mais l'accroissement de la complexité de ces systèmes déplace le problème, et leur apport en termes de performances peut se voir singulièrement réduit en cas d'inadéquation entre la configuration du système de cache et les conditions d'exécution. Cette configuration se révèle difficile à gérer, à cause d'une part de la multiplicité des réglages possibles, et d'autre part de la grande variabilité des conditions d'exécution. C'est pourquoi il est judicieux de construire des systèmes autonomes, capables de prendre des décisions de configuration par eux-mêmes.

Cette idée de configuration autonome est en fait un des aspects du principe d'adaptation, qui peut se décliner de différentes manières. Il importe dans un premier temps de définir la notion d'adaptation, et en particulier son objectif. En effet, l'adaptation dans son acception originelle est un processus observé dans les êtres vivants, qui vise à assurer la viabilité d'un organisme vivant. De la même manière, lorsque l'on applique cette idée au domaine informatique, on constate que l'adaptation a pour objectif d'assurer la viabilité des systèmes auxquels elle est appliquée. Cette propriété de viabilité peut être déclinée en objectifs plus précis selon la fonction du système considéré comme l'amélioration des performances ou de la robustesse du système. Pour répondre à cette variété d'objectifs, l'adaptation peut être mise en œuvre de différentes manières. Cette diversité de mise en pratique se retrouve dans les diverses dénominations que l'on peut constater dans les travaux s'y rapportant : adaptabilité, flexibilité, adaptativité, variabilité, configurabilité. Cependant, on peut identifier un certain nombre de propriétés communes à l'ensemble de ces approches, telles que le caractère dynamique de tout système mettant en œuvre l'adaptation, l'importance de l'observation des conditions d'exécution, prérequis à toute adaptation, ou encore l'application de mécanismes d'adaptation à différentes échelles et à différentes structures des applications.

Un des mécanismes les plus élémentaires de l'adaptation en informatique concerne la modification des comportements d'une application pour répondre aux conditions d'exécution courantes. Ce mécanisme est notamment utilisé dans les applications qui d'une part disposent d'un grand nombre de variantes algorithmiques pour fournir un même service, et d'autre part s'exécutent au sein d'un environnement dynamique.

Les caches web correspondent à cette caractérisation. D'une part, les multiples fonctionnalités que l'on peut y identifier disposent souvent de nombreuses variantes répondant à des exigences variées. D'autre part, leur environnement d'exécution faisant intervenir de nombreuses entités reliées par des réseaux est changeant, tout comme les ressources disponibles (capacité du réseau, capacité de stockage, etc). Enfin, les objectifs des caches web peuvent se décliner en plusieurs versions, suivant que l'on veut mettre l'accent sur les performances perçues par les utilisateurs ou bien les économies réalisées en termes d'utilisation de la ressource réseau. Il nous semble donc judicieux d'appliquer ce mécanisme élémentaire – l'adaptation des comportements – aux systèmes de caches.

Diverses techniques ont déjà été proposées pour mettre en œuvre ce type d'adaptation. On peut les retrouver dans des applications particulières conçues spécifiquement dans cet objectif, ou bien dans des approches plus abstraites visant à fournir des outils, comme des cadres de conception (*framework*), des langages ou des systèmes d'exploitation. On peut distinguer au sein de ces différentes techniques un modèle commun permettant notamment d'obtenir les différentes propriétés générales de l'adaptation que nous avons mentionnées précédemment. La compréhension de ces différentes techniques, ainsi que leur comparaison, permet de mieux cerner leurs apports et leur adéquation aux besoins.

L'objectif de cette thèse est d'extraire un modèle commun à ces différentes techniques d'adaptation de comportements, dans un double dessein. Tout d'abord, un tel modèle fournit une grille permettant d'évaluer et de comparer les outils adaptés au problème que l'on souhaite traiter. De plus, il expose les mécanismes fondamentaux de l'adaptation de comportements qui sont souvent difficiles à extraire d'une mise en œuvre particulière, à cause de l'infrastructure de réalisation qui est en place. La compréhension plus large du mécanisme, par l'exposition du paradigme de l'adaptation, contribue à la prise en compte d'aspects qui auraient pu être ignorés dans le cas de l'application directe d'une technique.

Pour atteindre cet objectif, nous avons tout d'abord clarifié le cadre général de l'adaptation en partant de l'objectif général de viabilité du système mentionné précédemment. Nous avons étudié les différentes déclinaisons de cet objectif et les différentes manières de le remplir, postulant l'existence d'un principe d'évolution des systèmes informatiques qui tendent vers de plus en plus d'autonomie. Cette phase nous a permis de mieux cerner le cadre plus précis de l'étude que nous désirons mener, l'adaptation dynamique de comportement. Le choix de ce cadre d'étude a été fixé par l'analyse que nous avons conduite auparavant sur le fonctionnement des caches web. Au vu des multiples techniques disponibles pour mettre en œuvre ce type

d'adaptation, nous avons voulu en présenter un modèle général. La formalisation que nous avons choisie pour cela est celle des patrons de conception qui indiquent, pour un contexte donné, un type de solution éprouvée déjà mise en œuvre dans d'autres systèmes, sans préjuger des moyens techniques utilisés pour la réaliser.

Ce document est structuré en cinq chapitres. Le premier chapitre est consacré à l'analyse du fonctionnement des caches web qui nous a conduit par la suite à nous intéresser au problème de l'adaptation. Après avoir mentionné les différentes propriétés attendues des caches web, nous présentons les différentes fonctionnalités qui sont mises en œuvre pour les réaliser, en insistant sur celles qui se révèlent propices à la mise en place de mécanismes autoadaptatifs. Notre étude de l'adaptation nous a conduit à constater l'importance de la collecte d'informations. Nous présentons donc les critères et outils de mesure qui peuvent être utilisés dans cette optique.

Le chapitre 2 vise à clarifier le domaine de l'adaptation considéré au sens large. Après avoir identifié la viabilité comme objectif principal du processus d'adaptation, nous en présentons les diverses déclinaisons et évoquons une loi générale d'évolution des systèmes informatiques qui en découle, indiquant que les systèmes vont vers une plus grande autonomie. Constatant la variété des termes relatifs à l'adaptation, nous nous intéresons à l'étude des critères permettant de positionner les différentes approches entre elles. Les deux principaux critères consistent en le type d'adaptation, défini par la structure du système soumise à l'adaptation (paramètre, comportement, architecture), ainsi que la localisation de l'adaptation dans un schéma temporel à deux axes relatif au cycle de vie du logiciel, intégrant d'un côté le moment d'injection du comportement et de l'autre côté le moment où l'adaptation est effectuée. Sur la base de ces critères, nous proposons une dénomination générique permettant de spécifier de manière plus précises le mode d'adaptation considéré. Nous appliquons cette dénomination à différents termes relatifs à l'adaptation, qui sont exprimés par le biais d'un vocabulaire spécifique : adaptabilité, flexibilité, adaptativité, variabilité, configurabilité. Cette classification nous permet de plus de préciser le cadre de l'étude que nous désirons mener dans le cadre de nos travaux sur les caches web, l'adaptation dynamique par le système de comportements existants.

Face au grand nombre de mécanismes permettant d'effectuer une telle adaptation, nous avons voulu en déterminer une architecture commune afin d'en faciliter la compréhension. Nous détaillons dans le chapitre 3 un patron de conception que l'on peut retrouver, parfois partiellement, dans différents systèmes. Nous présentons tout d'abord la notion de patron de conception, en revenant notamment sur son historique, son expression dans le domaine informatique et ses modes de représentation. Nous présentons ensuite la structure du patron de conception *Stratégie Autoadaptative* que nous avons identifié dans divers systèmes adaptatifs. Les composants de cette structure sont étroitement liés aux différentes étapes du processus d'adaptation. Nous étudions la manière dont le patron de conception *Stratégie Autoadaptative* permet de répondre aux besoins de deux types d'adaptation, *on action* et *on change*, puis nous précisons pour chacun des éléments les problématiques que l'on peut isoler dans leur mise en œuvre. Enfin, nous décrivons la manière dont les deux types d'adap-

tation traités peuvent être combinés, ainsi que la capacité du patron de conception de s'appliquer de manière récursive à certains de ses éléments.

Un patron de conception identifie un modèle de solution que l'on peut trouver dans plusieurs systèmes existants. Nous proposons dans le chapitre 4 de procéder à une analyse de systèmes existants au travers du filtre du patron de conception *Stratégie Autoadaptative*. Nous nous intéressons d'une part à des systèmes adaptatifs tels que des supports de stockage, des systèmes d'exploitation ou des protocoles. D'autre part, nous utilisons également le patron de conception pour analyser des outils permettant de réaliser des systèmes adaptatifs, afin d'identifier quels aspects y sont considérés, ce qui donne l'occasion de constater la variété des techniques disponibles : outils d'analyse, intergiciels (*middleware*), cadres de conception, langages dédiés, systèmes d'exploitation.

Le cinquième chapitre illustre la mise en œuvre de l'adaptation par les développements que nous avons effectués dans deux domaines. Le premier domaine est celui du tri adaptatif, qui nous a notamment permis d'identifier les deux types d'adaptation (*on change* et *on action*). Le second domaine est celui des caches web. Nous avons conduit des expérimentations autour de deux systèmes. Le premier est un simulateur de caches web, *Saperlipopette !*, au sein duquel nous avons mis en œuvre une stratégie de remplacement adaptative pour étudier les mécanismes mis en jeu. Le second système est la plate-forme de conception de services web Jigsaw, où nous avons mis en place une infrastructure appliquant l'adaptation au mode de stockage des données.

En conclusion, nous présentons la contribution que nous avons apportée, et donnons quelques perspectives de prolongation de ces travaux.

CHAPITRE 1

Les caches web

La popularité grandissante du WWW en a fait un outil de consommation courante, et l'on n'a pas encore atteint les limites de son expansion, si cela est concevable. L'infrastructure le soutenant (Internet) a jusqu'ici réussi à absorber l'augmentation des besoins, mais des problèmes de performance sont toujours à déplorer. Une des manières d'essayer d'alléger la charge des réseaux est d'introduire à des emplacement privilégiés des caches, équipements destinés à répliquer les données les plus populaires.

Cependant, l'administration de ces systèmes de cache est rendue ardue par leur complexité. Afin qu'ils puissent s'accommoder de conditions de fonctionnement et d'utilisation diverses, les caches utilisent des algorithmes variés pour accomplir leurs tâches internes. Cette grande variété algorithmique a pour pendant une complexification des mécanismes de configuration. De plus, le caractère fortement dynamique de l'environnement d'exécution des caches rend encore plus ardue la détermination de la configuration adéquate par un administrateur.

Il nous paraît alors intéressant d'amener les caches web vers une plus grande autonomie, qui leur permettrait de prendre certaines décisions de configuration par eux-mêmes et d'adapter ainsi leur fonctionnement aux conditions d'exécution. Dans cette perspective, nous cherchons à identifier les aspects des caches qui peuvent faire l'objet d'une telle autonomisation.

Dans un premier temps, nous rappelons les origines des systèmes de caches et préciser certains concepts principaux. Nous énonçons ensuite dans le paragraphe 1.4 certaines des exigences que l'on doit imposer à un cache, et décrivons successivement les intérêts et les inconvénients que présentent les caches pour différentes catégories de personnes (utilisateur final, fournisseur d'accès, fournisseur de service). Puis nous nous intéressons dans le paragraphe 1.7 aux diverses fonctionnalités identifiables dans un cache web, en nous focalisant plus précisément sur celles qui offrent les plus grandes potentialités d'adaptation. Dans l'optique d'une adaptation des caches, il est important de pouvoir observer et évaluer les conditions d'exécution ainsi que les performances du système. Nous présentons dans le paragraphe 1.8 les différents critères permettant de juger des performances, ainsi que des techniques permettant de les mesurer. Enfin, nous indiquons dans le paragraphe 1.9 les domaines les plus

propices à la mise en œuvre de mécanismes d'adaptation.

1.1 Introduction

1.1.1 Internet et World Wide Web - les débuts

Internet peut être défini comme un ensemble d'ordinateurs pouvant communiquer via des réseaux électroniques. D'abord outil de recherche universitaire, il a facilité la mise à disposition de nombreux services. Le plus populaire en est aujourd'hui le WWW, qui largement contribué à la découverte d'Internet par le grand public. Le WWW est en fait la description d'un moyen de transporter et de référencer de l'information. Il a été mis au point en 1990 [Cai95] par Tim Berners-Lee, alors consultant pour le CERN [CER], afin d'offrir aux chercheurs de cet établissement un moyen simple de diffuser les informations dont ils disposaient. Il consiste essentiellement en le protocole HTTP (pour *HyperText Transfer Protocol*) [FGM+97] et le format de documents HTML (pour *HyperText Markup Language* [W3Cd], dont le successeur est XML *Extensible Markup Language* [W3Cc]. Des logiciels spécifiques, appelés navigateurs, sont utilisés pour accéder à l'information.

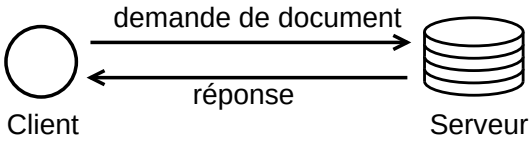

FIG. *1.1* — Le fonctionnement du WWW

Le fonctionnement du web est de type client-serveur, comme on le voit dans la figure 1.1. Un utilisateur, par le biais d'un logiciel de navigation tel que *netscape* ou *lynx* fonctionnant sur un ordinateur que l'on qualifie de client, peut accéder à des données situés sur un autre système informatique, que l'on qualifie de serveur. Pour ce faire, après l'établissement d'une connexion entre le client et le serveur, une requête est émise dans un format spécifié par le protocole HTTP [FGM+97]. Elle contient généralement un nom de ressource qui doit se trouver sur le serveur. Le serveur localise la ressource et renvoie son contenu en réponse au client.

La nature de la ressource concernée n'est pas fixée. Il est possible de transférer des documents au format texte aussi bien que des images ou du son. Cependant, un format particulier, appelé HTML, a été mis au point en parallèle avec le protocole HTTP. Le but de ce format est de structurer des documents afin de leur permettre de référencer d'autres documents, selon un schéma hypertexte.

Les standards HTTP et HTML ont évolué depuis leur conception, tout en maintenant une interopérabilité avec les versions antérieures. L'évolution de ces standards est supervisée par le le consortium W3 [W3Cb], dirigé par Tim Berners-Lee.

1.1.2 Évolution du trafic sur Internet

Depuis la diffusion du WWW et surtout la démocratisation de son accès, le trafic sur Internet a évolué de manière significative. Cette évolution concerne non seulement la quantité d'informations disponibles, mais également la nature des informations transmises.

Évolution de la quantité de trafic

Depuis le lancement du WWW, le nombre de serveurs Web n'a pas cessé de croître de manière exponentielle, comme le montre la figure 1.2. La quantité d'informations disponibles croît donc en proportion, ce qui encombre d'autant plus les connexions.

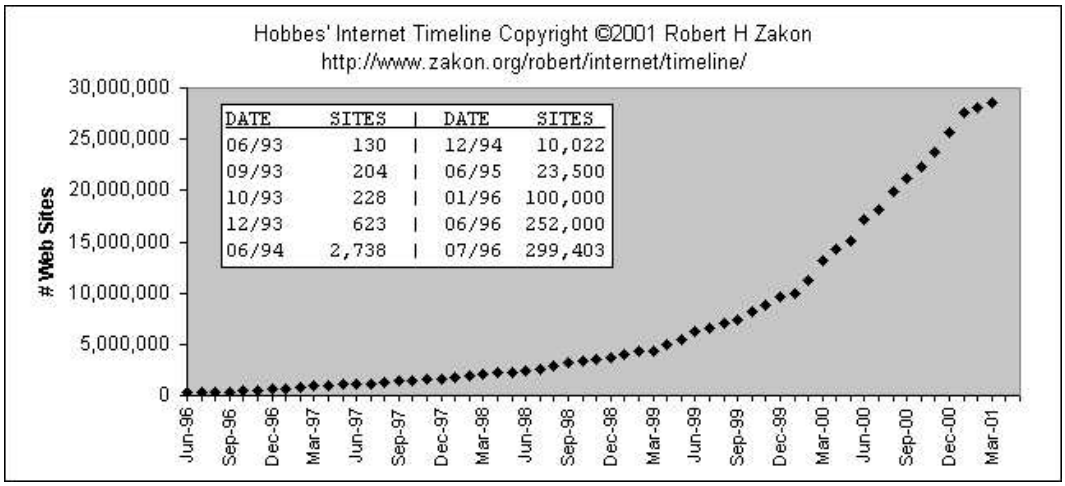

FIG. 1.2 — Croissance du trafic dédié au web

Le volume de trafic sur le réseau double chaque année, tandis que les coûts des équipements de communication et de routage ne baissent que de 30 % [MMV94].

Concernant plus particulièrement le trafic web, nous sommes passés d'un trafic quasiment insignifiant [Cai95, Gra97, TMW97] par rapport aux autres services à une saturation des liaisons disponibles.

Un autre problème posé par cette croissance est la recherche d'informations. De nombreux moteurs de recherche permettent de faire une recherche par mots-clés, mais ils ne peuvent maintenant plus indexer l'ensemble des sites existants [Gra97].

1.1.3 Évolution du type de trafic

Le trafic n'a pas seulement évolué en quantité : la nature des informations transportées a également significativement été modifiée.

Le passage du WWW d'outil de recherche à bien de consommation a entraîné une transformation du contenu des serveurs, passant de documents austères mais informatifs à des documents faisant intervenir de nombreux médias (image, son, vidéo). La taille des données a donc augmenté de la même manière que leur nombre, et cette évolution n'est pas achevée. L'importance du protocole HTTP est également croissante, proportionnellement aux autres protocoles utilisés sur Internet. En 1991, 44% du trafic sur Internet était redevable au protocole de transfert de fichiers *ftp*. En 1997, le protocole HTTP était à l'origine de 65 à 80% du trafic [TMW97].

Enfin, on assiste à une autre évolution qualitative, importante par rapport à la problématique des caches : les nouvelles applications du WWW ont de plus en plus besoin d'introduire des aspects transactionnels dans leur fonctionnement. Les protocoles existants n'ayant pas été conçus dans cette perspective, les concepteurs doivent avoir recours à divers artifices (*cookies*, codage d'adresse, etc.) qui ne sont pas forcément compatibles avec l'utilisation de caches. Cet enjeu reconnu commence à être pris en compte par certains projets de recherche, que l'on mentionne au paragraphe 1.6.2. De plus, de par le fonctionnement hypertexte du WWW, de nombreuses ressources (des images notamment) utilisées par les applications dynamiques du web constituent des éléments plus statiques pouvant profiter de l'utilisation d'un cache.

1.1.4 Enjeux actuels

Comme on vient de le voir, l'évolution en quantité et en qualité du trafic impose de nouvelles contraintes sur l'infrastructure servant à transporter l'information du WWW. Des solutions alternatives doivent être proposées. Le Consortium W3 [W3Cb] a donc mis en place de nombreux groupes de travail consacrés aux différents types de solutions : évolution du protocole HTTP, étude de la propagation et la réplication, etc.

Les systèmes de caches, dont le principe de fonctionnement est décrit au paragraphe 1.2, apportent au problème de diffusion à grande échelle de documents une solution intéressante bien qu'imparfaite, et que l'on peut rapidement mettre en œuvre. Les coûts induits par la mise en place d'un système de cache sont plus faibles que les économies qu'il permet de réaliser [dJ97, Mel97]. De plus, en répartissant la charge imposée aux serveurs, il facilitent le passage à l'échelle des infrastructures du WWW. D'autres projets tels que Globe [vSHT99] ont une approche du problème des environnements plus orientée autour de la notion d'objet, mais qui nécessite de déployer un système spécifique. Un des avantages de l'approche *proxy cache* est de fournir une réponse intégrable sans modifications à un environnement existant.

Nous nous intéressons donc plus particulièrement aux aspects de cache. Le docu-

ment [W3Ca] fournit une modélisation théorique du problème.

Nous allons voir dans un premier temps un rapide historique des principaux caches et produits associés. Puis nous décrivons un certain nombre de fonctionnalités des caches web, directes ou indirectes. Les caches web ne sont bien sûr pas une panacée, et ils apportent également leur lot d'inconvénients, que nous aborderons.

Nous traitons ensuite des différents aspects techniques mis en jeu dans les caches web, en détaillant pour chacun les perspectives de développement ainsi que la grande variété algorithmique présente dans chaque aspect. Nous étudions notamment le déploiement des caches web, plus précisément l'architecture et le dimensionnement, ainsi que les problèmes de politiques de remplacement.

Enfin, nous proposons un aperçu des critères d'évaluation des caches web et des techniques d'évaluation permettant de quantifier ces critères.

1.2 Concepts préliminaires

Un *proxy* est un programme intermédiaire qui fonctionne à la fois comme serveur et comme client afin d'effectuer des requêtes pour le compte d'autres clients. Les requêtes sont traitées de manière interne ou bien simplement transmises après une éventuelle modification à d'autres serveurs. Un *proxy* DOIT implémenter à la fois les aspects serveur et client. Un *proxy* transparent[1] est un *proxy* qui ne modifie pas la requête ou la réponse d'une autre manière que ce qui est nécessaire pour l'authentification et l'identification du *proxy*. Un *proxy* non transparent est un *proxy* qui modifie la requête ou la réponse afin de fournir des services supplémentaires aux applications, tels que la conversion de type d'information, la compression de protocole ou l'anonymisation des requêtes. À moins que le caractère transparent ou non du *proxy* soit explicitement précisé, les exigences des *proxy* HTTP s'appliquent aux deux catégories.

La figure 1.3 présente un exemple de *proxy* utilisé afin de permettre à des clients situés dans un réseau à accès protégé par un pare-feu (*firewall*) d'accéder à des serveurs web situés à l'extérieur du réseau local.

On désigne par cache web une zone de stockage de messages de réponses à des requêtes pour un programme, ainsi que le système qui contrôle le stockage, l'accès et la suppression des données. Un cache web stocke les données cachables[2] afin entre autres de réduire le temps de réponse et la charge réseau sur les requêtes ultérieures des mêmes documents. C'est une fonctionnalité spécifique ajoutée à un *proxy*. Dans la suite de ce document, nous utiliserons la dénomination *cache* pour désigner un

[1] Le terme *proxy transparent* se rapporte à un *proxy* sémantiquement transparent, comme décrit dans [FGM+97], et non à ce qui est classiquement compris dans la communauté des caches web, qui correspond plutôt au *proxy d'interception*.

[2] Une ressource est dite cachable si un cache est autorisé à en stocker une copie afin de l'utiliser lors du traitement de requêtes ultérieures. Même si une ressource est cachable, des contraintes supplémentaires peuvent empêcher l'utilisation de sa copie cachée pour une requête particulière.

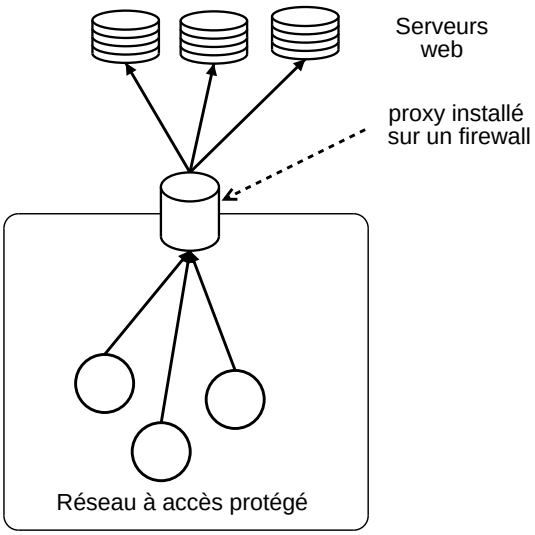

FIG. *1.3* — *Proxy* utilisé en tant que passerelle

cache web. Les acceptions plus générales du terme (caché mémoire, cache de système de fichiers, etc.) seront précisées explicitement.

Bien que pouvant fonctionner de manière autonome, un cache web est généralement amené à coopérer avec d'autres caches web, afin de constituer ce que l'on appelle un *système de cache*. Un système de cache est donc plus généralement une entité, constituée d'un ou plusieurs ordinateurs, remplissant de manière coordonnée la fonction de cache. Les éléments individuels du système de cache sont appelés les *nœuds*. Lorsque l'on veut désigner plus spécifiquement un mode de fonctionnement distribué entre plusieurs machines, on parle de cache distribué.

Nous appellerons *produit* une mise en œuvre spécifique d'un système de cache, qu'elle soit libre (Squid [Wes98b], Jigsaw [W3Ce], etc.) ou commerciale (Netscape [Net], Microsoft [Mic], Inktomi [Ink], etc.).

1.3 Historique des caches web

Le premier *proxy* déployé sur le WWW a été le serveur `httpd` du CERN [LA94]. Les fonctionnalités de passerelle ont été mises en œuvre pour divers protocoles dès la version 2.0, diffusée en mars 1993. Elles avaient principalement pour but de permettre à des utilisateurs situés derrière un pare-feu d'accéder aux serveurs web du réseau externe (voir figure 1.3).

Cependant est très vite apparu l'intérêt d'un tel point de passage unique. En effet, les différents clients à l'intérieur d'un réseau local sont susceptibles d'avoir des intérêts communs, et donc d'accéder à des documents identiques. L'ajout d'une fonctionnalité de cache permettant au *proxy* de conserver localement une copie des do-

cuments demandés et de restituer cette copie lors des requêtes similaires ultérieures, offre ainsi un moyen de pallier en partie les problèmes d'engorgement du réseau. La fonctionnalité de cache a donc été rajoutée dans la version 2.16 du serveur du CERN, en février 1994, afin d'améliorer les performances.

D'autres projets se sont alors intéressés au cache non comme une fonctionnalité supplémentaire ajoutée à un *proxy* simple, mais comme un produit à part entière. Harvest [BDH+95] est un projet initialement tourné vers l'indexation des documents. Au cours de son développement, des fonctions de cache et de réplication ont été ajoutées. Le projet a par la suite été abandonné, mais le développement de la partie cache s'est poursuivi pour donner naissance au cache Squid [Wes98b], qui reste un des produits de cache les plus utilisés ainsi qu'une plate-forme de développement de nouvelles fonctionnalités.

De nombreux acteurs commerciaux sont très vite arrivés sur ce terrain. Citons les plus importants Netscape [Net], Microsoft [Mic], Infolibria [Inf], CacheFlow [Cac99], Inktomi [Ink], Cisco [Cis]. On constate généralement une focalisation sur un aspect particulier : tenue en charge [Inf], facilité d'installation et de configuration [Net], fonctionnement transparent [Cis], préchargement des données [Cac99], etc.

D'autres acteurs ont proposé des services plus spécifiques, tels Akamai [KLL+97] qui fournit à ses clients non seulement un produit de cache, mais également une infrastructure dédiée de réplication et de routage des informations qu'ils désirent diffuser, assurant ainsi une bonne qualité d'accès aux utilisateurs.

Aujourd'hui, le domaine des caches web est très actif, tant dans la recherche académique que dans le monde industriel. Les performances des solutions proposées sont sans cesse croissantes, bénéficiant tant des innovations logicielles que matérielles. Il est difficile de distinguer la part des améliorations dues au matériel de celles dues au logiciel, à cause de l'intégration croissante matériel/logiciel de la plupart des produits de caches commerciaux.

1.4 Exigences

Le Consortium W3 [W3Cb] a énuméré un certains nombre des propriétés fondamentales que l'on doit attendre d'un système de cache [W3Cf]. Certains aspects font déjà l'objet de traitements importants dans des projets de recherche, tandis que d'autres sont moins souvent évoqués.

1.4.1 Correction

Les fonctionnalités du protocole HTTP/1.0 relatives au contrôle des copies des documents étaient peu nombreuses et peu efficaces. Les concepteurs d'applications devaient par exemple souvent désactiver le système de cache pour obtenir un fonctionnement correct. L'obtention rapide d'une copie légérement périmée peut convenir

pour de nombreuses applications, mais d'autres, comme le commerce électronique, ont besoin de s'assurer que les documents présentés à l'utilisateur sont à jour. On utilise le terme de *fraîcheur* (*freshness*) pour qualifier cette propriété des documents. Un système de cache doit donc être en mesure de fournir aux développeurs d'applications et de service un niveau de cohérence spécifié. Le protocole HTTP/1.1 [FGM+97] a notamment été étendu pour permettre la spécification précise du contrôle des caches à travers certaines directives.

1.4.2 Efficacité

On peut définir l'efficacité d'un système comme le rapport entre la charge d'un réseau de référence et la charge du réseau actuel, où la charge du réseau de référence est la somme du nombre d'octets transférés entre chaque nœud du réseau en utilisant un protocole HTTP sans cache. Cependant, cette mesure n'est pas forcément la plus pertinente. D'autres critères sont présentés dans le paragraphe 1.8.2.

1.4.3 Temps de réponse

Un aspect fondamental pour le WWW est la minimisation du temps d'accès à un document. Des études indiquent que la capacité de réflexion de l'esprit humain est dégradée lorsque les temps d'accès dépassent 100ms [CMN83]. Or les latences globales des réseaux sont habituellement de cet ordre. L'introduction de caches permet en un sens de rapprocher la source des informations de l'utilisateur, et par là de diminuer les temps de réponse. De plus, l'impact des transferts sur l'occupation des réseaux et la charge des serveurs est diminué, ce qui participe également à l'amélioration des performances du WWW. Le préchargement de documents susceptibles d'être consultés peut également contribuer à cette évolution, mais la consommation excessive de ressources du réseau par les documents préchargés inutilement incite à ne pas utiliser cette technique sans attention.

1.4.4 Limitations de stockage

On doit s'attendre à ce que chaque nœud constituant le système dispose d'une capacité de stockage limitée à consacrer aux données cachées et aux informations annexes – ou méta-informations – nécessaires au fonctionnement du système. Le système doit pouvoir fonctionner correctement en mode dégradé si l'on atteint les limites de stockage.

Une exigence supplémentaire concerne l'extensibilité (*scalability*). Étant donné que chaque nœud du système ne possède qu'une capacité de stockage limitée, une attention particulière doit être portée sur la possibilité d'extension – via la coopération et la distribution notamment – des systèmes.

1.4.5 Intervention humaine minimale

Les ressources de stockage, de traitement d'information ou de transfert sont mises en place par des humains. L'expérience avec des systèmes mis en place depuis longtemps sur Internet a montré que les utilisateurs finaux ou les administrateurs système ne doivent pas avoir à effectuer manuellement des opérations de routage ou d'optimisation de configuration dans des systèmes à grande échelle. Non seulement est-ce très coûteux en temps, mais de plus la nature dynamique de telles configurations rendrait rapidement obsolètes les configurations choisies. Il est donc important que la plupart des configurations soient totalement automatisées, bien que des décisions locales puissent influencer leur comportement. Il nous semble donc pertinent d'apporter aux systèmes de caches une plus grande autonomie de fonctionnement, par le biais de mécanismes autoadaptatifs.

1.5 Intérêts des caches web

Les apports des caches web sont multiples. Au delà de la simple accélération des transferts, on peut identifier un certain nombre des propriétés que l'on cherche à obtenir.

1.5.1 Intérêt pour l'utilisateur final

La mise en place d'un cache au sein d'un réseau réduit naturellement les temps d'accès des utilisateurs finaux aux objets situés sur des serveurs externes [Liu98, LAJF98, Woo96]. Cette réduction est la conséquence d'une part de la rapidité de service du cache, et d'autre part de l'économie de bande passante sur les liens externes.

En effet, les documents présents dans un cache sont servis quasi-immédiatement. Ce temps de service peut être décomposé entre temps de latence et temps de transfert.

Le temps de latence représente le temps entre l'initiation d'un transfert, qui se fait classiquement par l'activation d'un lien dans un navigateur, et le début de la réception du document. Le mécanisme d'établissement de connexion de TCP participe pour une bonne part à ce temps d'établissement [CDF+98]. Comme il a été mentionné dans le paragraphe 1.4.3, le temps d'accès aux informations ne devrait idéalement pas dépasser 100ms. Il est donc important de réduire cette latence au maximum.

Le temps de transfert représente le temps total de transfert du document. Dans le cas d'un document de taille importante, il est largement supérieur au temps de latence. Pour les documents de taille plus réduite, la différence est moindre. Dans tous les cas, le temps de transfert est relatif à la taille du document.

La notion de performance évaluée à l'aune temporelle peut donc être étudiée à différentes échelles de précision. Les temps de latence participent à la perception d'un

temps de réaction, tandis que les temps de transfert sont plus pénalisants pour les utilisateurs manipulant des données de taille moyenne à importante. Il faut également considérer que les durées mesurables présentées ci-dessus ne permettent pas de qualifier totalement le degré de satisfaction d'un utilisateur relativement au temps. D'autres facteurs plus subjectifs entrent en compte, comme la vitesse d'affichage du document ou la valeur d'utilité associée au document[Joh97].

Par ailleurs, le transfert des documents non-cachés bénéficie de la bande passante économisée. En réduisant la charge sur les liaisons extérieures, les caches web augmentent la bande passante disponible pour les documents non-cachables ainsi que pour toutes les autres applications transférant des données via le réseau.

L'apport des caches web ne se limite pas à l'amélioration des performances. Leur position privilégiée permet d'offrir à l'utilisateur final un certain nombre de fonctionnalités, comme par exemple la possibilité d'anonymiser les connexions, et d'assurer ainsi un certain niveau de protection de ses données individuelles.

1.5.2 Intérêt pour le fournisseur d'accès

Un des objectifs principaux d'un système de cache est d'obtenir une diminution de l'utilisation de la bande passante, donc une diminution du trafic et une réduction de la congestion du réseau. Outre une amélioration des performances, cette réduction du trafic permet de limiter les besoins en bande passante, et donc le coût des équipements nécessaires [dJ97, Mel97].

De plus, un fournisseur d'accès peut profiter du passage systématique des informations à travers un système central pour appliquer aux données des traitements divers. Une utilisation intéressante est par exemple l'application systématique d'antivirus aux exécutables téléchargés, la suppression des codes *javascript* ou *java* pouvant poser des problèmes de sécurité à l'intérieur d'un réseau, ou encore la suppression automatique des bannières de publicité [Jun].

Enfin, ce point de passage privilégié offre la possibilité d'analyser les modèles d'utilisation du web afin notamment de constater l'évolution des demandes des utilisateurs et déployer des équipements en conséquence [AW96].

1.5.3 Intérêt pour le fournisseur de service

La prise en charge d'une partie du trafic par les caches web, notamment en ce qui concerne les documents les plus populaires, apporte une réduction de la charge imposée aux serveurs. On obtient ainsi une répartition de la charge par diffusion de l'information.

Cette propriété est explicitement exploitée dans le cas des *caches inverse* [WDR00], placés non plus au plus près des utilisateurs pour leur permettre d'accéder à de nombreux serveurs, mais placés au plus près d'un serveur web pour lui permettre de servir un nombre de documents plus élevé.

Un avantage supplémentaire concerne l'augmentation de la robustesse du service, notamment en permettant l'accès à des documents dont le serveur original est inaccessible. En cas de défaillance d'un serveur ou d'une des liaisons permettant d'accéder à ce serveur, les copies cachées peuvent être toujours disponibles, cachant parfois à l'utilisateur le problème survenu à la source.

1.6 Inconvénients des caches web

Les caches web ne sont pas une panacée et ils apportent également leur lot d'inconvénients concernant la mise-à-jour des documents, les performances ou encore le droit d'auteur.

1.6.1 Des documents pas toujours à jour

Des mécanismes pour assurer la cohérence des données cachées sont mis en place dans tout système de cache. Néanmoins, ces mécanismes ne garantissent pas à tout instant que la copie cachée est à jour. La qualité de la *fraîcheur du document* peut être estimée au mieux par des méthodes statistiques, ou bien des mécanismes en cours d'intégration dans les protocoles. HTTP/1.1 [FGM+97] permet en particulier d'obtenir des informations sur la fraîcheur des informations consultées. Nous reviendrons sur cet aspect dans le paragraphe 1.7.6.

1.6.2 Un fonctionnement pas toujours transparent

Il est impossible de fournir un *service totalement transparent* à l'utilisateur. En particulier, le résultat de la consultation de sites *ftp* ou encore les messages d'erreurs [Cor98] ont un aspect différent lors de l'utilisation d'un cache, ce qui peut perturber l'utilisateur, et rendre le diagnostic de problèmes plus difficile. Il faut donc essayer de rendre les caches les plus transparents possible, au sens précisé dans le paragraphe 1.2.

1.6.3 Des performances pas toujours au rendez-vous

Les traitements supplémentaires effectués par un système de cache peuvent augmenter les temps d'accès à l'information, en particulier en cas de non-présence dans le cache du document demandé. Un des objectifs d'un système de cache est donc de maximiser le taux de succès (*hit-rate*) et de minimiser le coût de traitement dans le cas de non-présence dans le cache. Ce paramètre est d'autant plus important dans un contexte d'architecture hiérarchique de caches, où les différents niveaux contribuent chacun au temps d'accès total.

Par ailleurs, un sous-dimensionnement du système de cache peut conduire à une diminution significative des performances par rapport au système initial sans cache, ce qui gênera l'utilisateur et le conduira à court-circuiter ce mécanisme, réduisant

à néant les efforts entrepris. Il est donc important, comme l'indique [Cor98], que l'utilisateur aie bien conscience du bénéfice du cache, tant au niveau individuel qu'au niveau d'une communauté d'utilisateurs.

Un autre inconvénient concerne le goulot d'étranglement et le point de panne unique que constitue un *proxy*. Il faut donc limiter le nombre de clients que celui-ci peut satisfaire, ou mettre en place des mécanismes de répartition de charge par différents moyens comme des commutateurs à équilibrage de charge, un serveur DNS tournant, des caches distribués, ou un produit tel que le *superproxy script* [SHA00].

Les scripts de configuration automatique (*PAC*) fournissent un moyen de contourner le problème de panne du serveur, en redirigeant les requêtes vers d'autres caches ou en s'adressant directement aux serveurs d'origine si nécessaire.

Les techniques de personnalisation des sessions (*cookies* [FCD+99]) ainsi que la génération automatique de documents [CDF+98, CZB98] réduisent l'intérêt des caches. Cependant, une étude récente [FCD+99] pratiquée sur un large panel d'utilisateurs durant plusieurs mois a montré que 8% seulement des requêtes n'étaient pas cachables. De plus, [KW98] indique, sur la base d'une étude de cas, que la plupart des réponses utilisant des *cookies* peuvent être cachées.

Enfin, des approches nouvelles ont été proposées pour tenter de gérer le problème des documents dynamiques. Le principe d'*Active Cache* [CZB98] par exemple est d'associer aux documents dynamiques des *applets*, et de les faire exécuter par les caches lorsqu'ils utilisent une version cachée du document, afin d'effectuer les traitements dynamiques sans avoir besoin de recontacter le serveur web.

1.6.4 Des biais dans les mesures

L'utilisation généralisée des caches fausse les statistiques d'accès aux sites web. De nombreux sites désirant avoir des statistiques plus précises décident de rendre leurs documents non-cachables, ce qui va à l'encontre d'intérêts plus généraux [Pit97]. Des solutions [ML97] ont été proposées pour tenter d'obvier à ce facteur, par l'ajout par exemple d'un entête *Meter* au protocole de transfert, qui permettrait aux caches d'informer des serveurs sur le nombre de consultation d'un document.

1.6.5 Des problèmes juridiques

En contrepoint à l'argument de protection de la vie privée, un cache fournit un moyen simple d'analyser les accès à Internet [AW96]. Il est donc possible pour un administrateur de cache d'espionner les utilisateurs. Les implications sont de même nature que dans le cas du courrier électronique ou des écoutes téléphoniques : tout dépend du degré de confiance accordé à chaque niveau du réseau. La conservation de traces d'accès [KMM97] aux systèmes de cache, et leur diffusion, pose ces problèmes de confidentialité de manière accrue.

En outre, le fait de conserver une copie des documents pose des problèmes légaux

vis-à-vis de la notion de *copyright* [CMT01]. Par exemple, en mars 1999, le parlement européen a voulu émettre une directive visant à interdire l'usage des caches à cause de cela. Cette directive n'a pas été mise en œuvre, mais l'anecdote souligne la pertinence de la question.

1.7 Fonctionnalités des caches web

Les caches web mettent en œuvre plusieurs fonctionnalités parmi lesquelles on peut distinguer [BP97, Pie99, Wan99, BP99] :

- le filtrage, qui inclut le contrôle d'accès et la détermination de la cachabilité des documents ;
- le préchargement de documents ;
- la coopération entre caches, qui se décline en architecture, partage d'informations entre caches, routage des informations ;
- la gestion de la cohérence des informations cachées ;
- le remplacement des documents dans un cache, qui oblige à fournir des estimations de la valeur des documents [Rou97] ;
- l'accès aux ressources, et en particulier le stockage des données.

Après un rapide parallèle avec les mécanismes de cache des systèmes de gestion de mémoire, nous allons détailler chacun de ces points.

1.7.1 Un parallèle

Il est intéressant de faire le parallèle entre les caches web et les mécanismes de cache plus connus appliqués aux systèmes de gestion de mémoire. Cependant, comme le rappellent Cao et Irani [CI97], les conditions d'applications peuvent être sensiblement différentes, particulièrement sur trois points.

Tout d'abord, une particularité notable des caches web est la non-uniformité des tailles des éléments cachés. Dans un cache mémoire, les lignes de caches sont de même dimension, ce qui peut amener des simplifications dans le fonctionnement. Les tailles des documents web sont très hétérogènes et peuvent varier de quelques octets à des centaines de méga-octets, en particulier avec le développement de nouvelles utilisations telles que la diffusion de vidéo ou d'autres contenus multimédias.

De plus, le temps de chargement des documents dans un cache n'est pas constant, même pour des documents de même taille. Il est possible de prendre en compte ce paramètre dans la gestion de la cohérence des données du cache. Les temps de transfert de données sont beaucoup plus homogènes dans les systèmes de gestion de mémoire, ce qui apporte des simplifications.

Enfin, les motifs d'accès aux données dans les caches mémoire sont différents de ceux qu'on peut trouver dans les caches web. Le mode d'accès aux documents web est en effet majoritairement en lecture seule, contrairement aux accès mémoire. De plus,

le trafic web est plus chaotique, mettant en jeu des parcours déjà peu ordonnés de plusieurs dizaines, voire centaines, d'utilisateurs. Les propriétés de localité temporelle et spatiale fortement présentes dans les systèmes de gestion de la mémoire le sont également dans les accès web, mais à un degré moindre.

1.7.2 Le filtrage

Le contrôle d'accès mis en œuvre au niveau des caches web permet de sélectionner de manière plus ou moins fine les systèmes ou utilisateurs pouvant bénéficier du système de cache, ainsi que les sites qu'il est permis de consulter.

Le cache doit également vérifier que le document qu'il télécharge pour le compte d'un client est cachable, i.e. qu'il a la possibilité d'en conserver une copie. Plusieurs facteurs peuvent réduire la cachabilité d'un document. Les documents générés dynamiquement ou ceux comprenant des restrictions d'accès ne sont pas forcément cachables. De même, un serveur peut explicitement demander aux caches de ne pas garder de copie de certains documents, via des entêtes spécifiques ajoutés à la réponse.

Contrôle d'accès

Les contrôles les plus simples contrôlent l'adresse IP de la machine cliente. Il est ainsi possible de restreindre l'accès à un certain nombre de machines du réseau interne, ainsi que d'interdire l'utilisation du cache aux machines externes au réseau.

Le contrôle par adresse étant limité, des systèmes plus élaborés ont été développés, qui utilisent des informations supplémentaires pour effectuer le contrôle. Squid par exemple propose un système de gestion de listes de contrôle d'accès extrêmement souple, permettant une gestion fine utilisant non seulement des paramètres des clients (adresse IP, identification de l'utilisateur, type de navigateur), mais également d'autres paramètres tels que des caractéristiques de l'URL demandée ou la date courante.

Dans Squid, une extension du contrôle d'accès, nommée *delay pools*, permet de limiter les vitesses de chargement suivant divers critères d'identification de classes d'utilisateurs, afin de mieux utiliser la bande passante disponible.

1.7.3 Le préchargement de données

Comme il a été mentionné précédemment, les premiers systèmes de cache mémorisaient les documents lors d'un premier accès par un utilisateur afin de réduire les temps de transfert lors des accès suivants. Des simulations fondées sur une hypothèse de taille de cache infinie [ASA+96] ont montré que le taux de réussite maximum d'un cache web ne pouvait excéder 50%. En d'autres termes, l'intérêt du travail autour

des politiques de remplacement ou de routage des données ne peut guère espérer pouvoir concerner plus d'un document sur deux.

Une technique inspirée des techniques déjà existantes dans le domaine des caches mémoire peut améliorer ce taux de réussite [KTP+96]. Il s'agit d'anticiper les requêtes futures et de précharger dans le cache les documents susceptibles d'être demandés.

Cao *et al.* [FCLJ99] identifient trois variantes dans la mise en œuvre de mécanismes de préchargement. Il peut être effectué par les clients pour les documents provenant des serveurs, par les caches pour les documents provenant des serveurs, ou par les clients pour les documents provenant des caches.

Les premières études ayant été effectuées à une époque où l'utilisation des caches était peu répandue, le préchargement était effectué par les clients sur des documents provenant des serveurs. Padmanabhan et Mogul [PM96] ont étudié les réductions de latence ainsi que l'utilisation du réseau en se basant sur des historiques d'accès à des serveurs web. Ils ont montré qu'un mécanisme de préchargement pouvait réduire la latence d'un facteur de 45%, au prix d'un doublement de la bande passante utilisée. De même, Chinen et Yamaguchi [iCY97] ont mesuré le taux de réussite maximum, qui passe de 40 à plus de 60%.

Cao *et al.* [FCLJ99] ont montré l'intérêt d'effectuer le préchargement des documents au niveau du cache. En effet, si l'on considère que les utilisateurs d'un cache partagent des intérêts communs, la position d'observateur privilégié lui permet de plus facilement prédire les accès futurs puisqu'il peut se baser sur le modèle d'accès des autres utilisateurs.

Kroeger, Long et Mogul [KLM97b] ont étudié plus précisément le préchargement au niveau des caches, en se focalisant plus particulièrement sur les améliorations en termes de temps de latence. Leurs mesures indiquent que la réduction du temps de latence, dans le cas d'un cache web classique, est de 26% sur leur ensemble de mesures, et qu'elle peut atteindre 41% avec l'ajout d'une politique de préchargement simple consistant à précharger systématiquement la totalité des liens associés au document courant. La prise en compte d'informations émanant du serveur, qui peut également identifier des stratégies de parcours récurrentes, améliore encore les performances.

Maltzahn *et al.* [MRGM99] utilisent le préchargement dans l'objectif d'optimiser l'utilisation de la bande passante disponible. Les documents susceptibles d'être requis sont donc si possible téléchargés durant les périodes creuses d'utilisation du réseau, afin de diminuer l'amplitude du pic d'utilisation suivant.

Cao *et al.* [FCLJ99] ont étudié l'intérêt du préchargement par les clients des documents provenant du cache, dans le cadre d'une connexion finale à faible débit telle qu'un *modem*. Leur étude montre que cette méthode, combinée avec une compression éventuelle des données, peut réduire la latence observée par l'utilisateur de 23%.

1.7.4 Architecture et dimensionnement

Les systèmes de caches web faisant intervenir plus de deux nœuds peuvent être organisés suivant divers schémas. On distingue habituellement l'organisation hiérarchique et l'organisation distribuée, qui sont généralement combinés en une architecture hybride. De plus, le dimensionnement du système est un paramètre important au regard des performances.

Architecture hiérarchique

Danzig, Hall et Schwartz [DHS93] ont étudié en 1993 le problème de l'augmentation du trafic des serveurs *ftp* et ont proposé une architecture hiérarchique répondant à un certain nombre des problèmes posés, tout en précisant que leur étude s'appliquait à d'autres protocoles. Les pionniers des caches hiérarchiques ont utilisé ces travaux pour construire leur système. L'organisation hiérarchique de caches web a été initialement proposée par le projet Harvest [BDH+95], ancêtre de Squid, et est largement déployée actuellement [RSB99, DMF97, dC01, Flo96]. Elle consiste en l'établissement d'une structure arborescente de caches, où un élément peut coopérer avec les éléments du niveau supérieur.

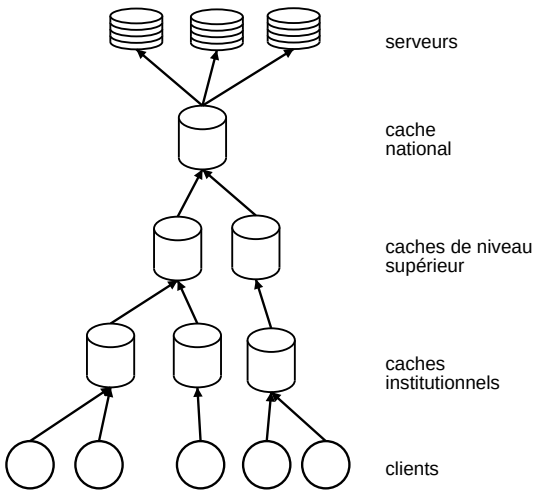

FIG. *1.4* — Architecture hiérarchique

Une répartition classique, représentée dans la figure 1.4, consiste à définir quatre niveaux [RSB99] : client, institutionnel, régional et national. Cette répartition, bien qu'apparemment géographique, est basée sur les différentes capacités des lignes d'interconnexion des divers niveaux. Au niveau client, on trouve le cache local intégré dans tout navigateur.

L'approche hiérarchique consiste à permettre à un cache d'un certain niveau d'interroger, en cas de non-présence d'un document demandé, un cache de niveau supérieur ou bien les caches de même niveau. Les caches de niveau supérieur n'interrogent

pas les caches fils. Ainsi, lorsqu'une requête n'est pas satisfaite par le cache client, elle est transmise successivement aux caches institutionnel, régional puis national. Si le cache de plus haut niveau ne contient pas de copie du document, il contacte le serveur original. Une fois que le document a été trouvé, il parcourt le chemin inverse jusqu'au client, laissant une copie dans chacun des caches qu'il traverse.

Cette méthode entraîne une sorte de filtrage de l'information, dans le sens où les informations les plus demandées se répandent au fur et à mesure dans l'arborescence, en partant du niveau le plus haut.

Les conséquences de l'utilisation d'une structure hiérarchique sont principalement une utilisation plus réduite de la bande passante, avantageuse notamment pour les sites ne disposant que d'une bande passante réduite, et une réduction du temps de connexion [RSB99, DMF97]. De plus, les temps d'établissement d'établissement des connexions sont réduits, de par la plus grande proximité des caches de bas niveau.

Enfin, on apporte aux clients un taux de réussite plus élevé, combinant les taux de réussite des différents niveaux. Duska, Marwood et Feeley [DMF97] ont observé, sur une configuration particulière, un taux de réussite passant de 45% pour un seul niveau de cache à 55% lors de l'ajout d'un cache de niveau supérieur.

Cependant, l'efficacité d'une architecture hiérarchique de caches peut être remise en cause par des problèmes de profondeur de hiérarchie, ainsi que par le dimensionnement des caches de plus haut niveau. En effet, chaque niveau de hiérarchie entraîne des délais dans l'acheminement des documents. Il est ainsi conseillé de ne pas dépasser quatre niveaux de caches [BBM+97], en comptant le cache local du navigateur, si l'on ne veut pas détériorer les latences observées.

Par ailleurs, les caches de plus haut niveau ont besoin d'une capacité de stockage plus importante à cause du nombre d'utilisateurs pouvant potentiellement accéder aux données. De plus, leur taux de réussite est moins élevé puisque la majorité des requêtes ont été satisfaites par les caches de niveau inférieur. Le trafic réseau est plus grand proportionnellement au nombre potentiel d'utilisateurs, augmentant encore la charge sur les caches de haut niveau. Selon une étude menée dans [DMF97], les taux de réussite pour les caches régionaux sont compris entre 24 à 45%. Pour des caches de troisième niveau, on atteint 19%. On peut également noter que le projet de cache national Renater [CG00], en France, a été abandonné en août 2000 après quelques années de service, à cause de la charge trop importante imposée sur le serveur et des gains jugés trop faibles en retour, notamment au regard des coûts d'équipement et d'administration ainsi que de la capacité croissante des accès internationaux de Renater.

Architecture distribuée

Une architecture totalement distribuée [Pov95, TDVK98] tente de mettre à plat l'espace de coopération entre caches web. On observe donc généralement une structure symétrique, où chaque nœud dispose des mêmes capacités que ses voisins. Cette

coopération peut s'effectuer de différentes manières, liées au protocole de distribution utilisé. Dans une structure purement distribuée, tous les caches sont considérés comme étant au même niveau, comme le présente la figure 1.5.

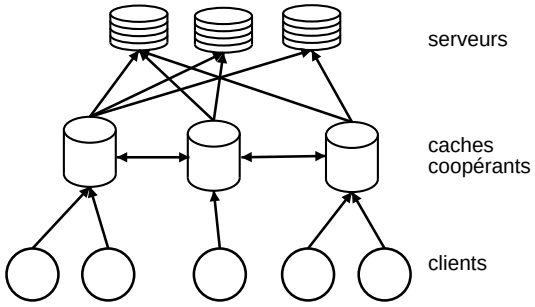

FIG. *1.5* — Architecture distribuée

Dans ce contexte, la plus grande partie du trafic utilise les liaisons les plus près des clients, évitant ainsi la congestion et les saturations de disques des niveaux intermédiaires. Les temps de transfert sont alors améliorés grâce à une utilisation plus équilibrée des différentes ressources réseau. Néanmoins, le passage à une échelle importante pose de nombreux problèmes, comme par exemple des temps de connexion plus importants et des problèmes administratifs relatifs à la coopération des caches. Cette technique est cependant adaptée à des échelles raisonnables.

Les architectures hybrides

Les deux types d'architecture, hiérarchique et distribuée, sont généralement combinés en une architecture hybride, afin de profiter de leurs avantages respectifs : le temps de réaction lors de l'établissement d'une connexion (la latence) est réduit dans une architecture hiérarchique, alors qu'une architecture distribuée diminue les temps de transmission, en évitant d'utiliser des liens trop congestionnés [RSB99].

Floyd propose dans le projet *Adaptive Web Caching* [Flo96, BO00] de considérer les systèmes de caches comme une manière d'optimiser la dissémination globale des données. Le projet se focalise sur le problème de la popularité soudaine de certains documents, qui peut apparaître en une durée très réduite. Le système *Adaptive Web Caching* consiste en de nombreux caches distribués organisés en groupes – qualifiés de mailles (*caching meshes*). Chaque cache peut dynamiquement intégrer ou quitter un groupe. Les caches communiquent à l'intérieur des groupes en utilisant une technique de multi-diffusion (*multicast*). La communication entre groupes s'effectue via les caches appartenant simultanément à plusieurs groupes.

Lorsqu'un document demandé est disponibles sur un des caches, ce dernier renvoie une copie non seulement à l'entité qui en a fait la demande, mais le diffuse également à l'ensemble du groupe dont il fait partie. Ainsi, plus un document est demandé à l'intérieur du système d'*Adaptive Web Caching*, plus il est répliqué dans chacun des groupes.

Adaptive Web Caching définit un protocole, le *Cache Group Management Protocol* (CGMP), qui permet de définir l'organisation des groupes, et les ajouts ou suppressions de caches à un groupe. Des techniques de vote sont utilisées à l'intérieur de chaque groupe afin de déterminer si un candidat peut être ajouté ou supprimé.

Choix d'architecture

L'architecture optimale étant spécifique à chaque site (topologie du réseau, comportements et exigences des utilisateurs, etc.), il est nécessaire de disposer d'outils ou de méthodes permettant de déterminer la configuration la plus pertinente.

Rodriguez, Spanner et Biersack [RSB99] ont ainsi construit un modèle analytique afin d'étudier les différents modes de coopération (hiérarchique, distribué, hybride) de caches dans un réseau, en vue de déterminer une répartition optimale des caches. Leur approche favorise l'établissement d'une hiérarchie de groupes de caches distribués, permettant de cumuler les avantages des deux modes.

Guillaume Pierre [Pie99] propose d'identifier des groupements d'utilisateurs à l'intérieur du réseau sur lequel doit être déployé le système de cache, via l'étude des historiques d'accès. Partant de cette étude, il propose diverses architectures en prenant en compte les tailles et l'organisation des communautés. Le fonctionnement de chaque architecture est ensuite simulé grâce au logiciel *Saperlipopette !*, qui permet dans un premier temps de choisir une architecture, puis de procéder à l'optimisation des tailles des différents caches de l'architecture choisie. L'étude de cas qu'il propose sur le réseau de l'INRIA est particulièrement parlante.

Dimensionnement

Que l'on soit dans un contexte de cache fonctionnant seul ou dans le cas d'une hiérarchie de caches coopérant entre eux, le problème du dimensionnement est crucial. Un fournisseur d'accès utilise un cache entre autres pour économiser des ressources. En cas de surdimensionnement, le coût d'installation et de maintenance du cache peut être supérieur au coût des ressources économisées. En cas de sous-dimensionnement, le cache risque d'introduire des pertes de performance au lieu d'améliorer ces dernières [BH96].

Les critères de dimensionnement du cache sont principalement la taille de l'espace de stockage alloué au système, ainsi que la capacité des réseaux connectant le cache d'une part à ses clients et d'autre part au reste du réseau, comprenant serveurs web et autres caches coopérants.

Guillaume Pierre [Pie99] propose d'utiliser un simulateur de caches (*Saperlipopette !*), et y intègre un optimiseur qui permet de déterminer, pour une architecture donnée, les tailles optimales des différents caches.

Kant et Won [KW99a] descendent jusqu'au niveau matériel et tentent d'analyser, au regard d'un trafic web donné, les besoins en termes de bande passante matérielle

(capacité des interfaces réseaux, du bus PCI, du bus mémoire, des disques, etc.). Rousskov et Soloviev [RS99] ont effectué une étude conduisant à des analyses similaires.

Kelly et Reeves [KR01] proposent deux approches analytiques permettant de déterminer la taille optimale d'un cache web. Ils s'intéressent plus particulièrement à donner une limite en termes de rentabilité aux systèmes de caches (donc une borne supérieure à leur capacité), en soulignant le fait que les besoins en performance sont orthogonaux.

1.7.5 Les protocoles de coopération

L'objectif des protocoles de coopération est de faire savoir à un cache si un autre nœud du système possède une copie du document demandé par un utilisateur. Ils peuvent être classés selon qu'ils tentent de localiser l'information *a posteriori* (ICP [Wes98a, WC98, WC97], HTCP [VW], CRISP [GRC97], Relais [MB97]) ou *a priori* (CARP [VR97], Digest [RW98, FCAB98]).

Localisation de l'information *a posteriori*

Le principe de cette catégorie de protocoles est de chercher à avoir une information sur le contenu des autres caches au moment où la requête pour un document a été reçue. Pour ce faire, le cache traitant la requête envoie une demande aux autres nœuds du système afin de déterminer si un de ces derniers contient une copie ou non.

ICP ICP [Wes98a, WC98, WC97] (*Internet Cache Protocol*) est un des premiers protocoles de coopération entre caches web. Élaboré initialement au sein du projet Squid [Wes98b], il a été mis en œuvre par la suite dans de nombreux produits libres (Harvest) ou commerciaux (Netcache, Netscape Proxy Server, etc).

ICP est utilisé pour une communication inter-caches. Un cache peut interroger ses voisins ou parents afin de déterminer si un objet y est présent.

ICP est basé sur UDP. Comme UDP est un protocole sans garantie d'intégrité, il est possible d'estimer les performances et l'encombrement d'un réseau à partir du taux de pertes de paquets ICP. Cette méthode, ainsi que la mesure du temps de transfert des informations, fournit une méthode d'équilibrage de charge pour les caches.

Cependant, on peut aboutir à l'apparition de faux *hits* (un objet présent dans un cache, mais inaccessible depuis un cache voisin par exemple). En effet, les caches demandent la localisation des objets par ICP puis les chargent par HTTP, mais ICP ne transporte pas d'informations à propos des entêtes HTTP associés à un objet. Ces entêtes peuvent contenir des informations de contrôle d'accès ou des directives à destination des caches, notamment sur la portée publique ou privée d'une ressource.

Il souffre également d'un problème de redimensionnement (*scalability*), de sécurité [WC98], ainsi que de performances à cause du délai qu'il introduit [RW98] lors de la phase de localisation d'un document.

HTCP HTCP [VW] est un protocole permettant de découvrir des caches HTTP ainsi que des données cachées, de gérer des ensembles de caches HTTP et de surveiller l'activité des caches.

Les entêtes HTTP sont des informations vitales pour les caches web. HTCP [VW] apporte la possibilité de les exploiter, ce qu'ICP ne permet pas. De plus, HTCP fournit des fonctionnalités de gestion de cache, offrant ainsi un moyen de contrôler l'activité d'un cache distant, et d'envoyer des indications sur des documents telles que l'emplacement de copies ou la cachabilité.

Dérivé d'ICP, il est également utilisé pour une coopération sur un mode distribué ainsi que sur un mode hiérarchique.

CRISP Les caches CRISP [GRC97] sont structurés comme un ensemble de caches autonomes, partageant leurs répertoires de cache à travers un service de localisation commun qui peut être interrogé par un message unique (voir figure 1.6).

Les serveurs individuels peuvent être configurés pour répliquer tout ou partie de l'ensemble des données, afin d'équilibrer les coûts d'accès, la surcharge du réseau et le taux de réussite, en fonction de la taille et de la répartition topographique des différents caches composant le système.

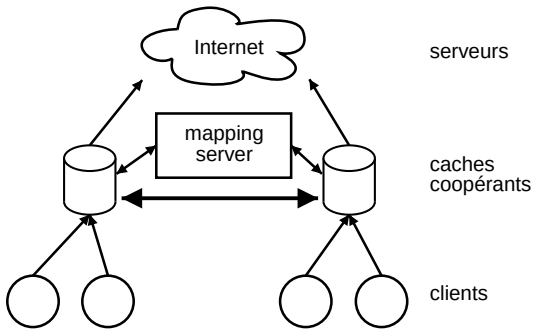

FIG. *1.6* — Architecture de CRISP

Relais

Relais [MB97] est un protocole de coopération développé à l'INRIA. Il garantit aux utilisateurs une progression monotone et rapide dans les versions des documents : dès qu'un cache apprend qu'un document est périmé, Relais se charge de rafraîchir les versions du document stockées par les partenaires. Pour des raisons de performance, toutes les actions de maintien de cohérence sont effectuées en tâche de fond.

Relais localise efficacement les documents : l'opération nécessite uniquement un accès à un répertoire local. Relais a été implémenté au-dessus de Squid, et est utilisé au sein de l'équipe de recherche SOR depuis 1997.

Localisation de l'information *a priori*

Dans ce mode de coopération, qui peut être complémentaire du premier, le système de cache peut localiser plus ou moins précisément le nœud du système le plus susceptible de contenir une copie d'un document.

CARP CARP [VR97] n'est pas un protocole de communication à proprement parler. Un des buts de CARP est d'éviter la communication inter-caches. Pour ce faire, il introduit une fonction de hachage distribuant l'espace des URL vers un ensemble de caches ainsi qu'une liste des caches appartenant au groupe (*Proxy Array Membership Table*), et des moyens de transmettre ces informations. Un agent HTTP (un client ou un *proxy*) implémentant CARP peut router intelligemment les requêtes pour une URL donnée vers un membre du groupe de caches (*Proxy Array*). Grâce à ce tri préliminaire, on évite de dupliquer l'information dans le groupe de cache, et on augmente les taux de *hits*.

CARP est utilisé pour des grappes de caches web et les clients peuvent utiliser CARP directement pour sélectionner l'élément du groupe le plus adapté. Un fonctionnement analogue est proposé par le *Super Proxy Script* [SHA00] de Sharp.

Digests Les résumés de contenus (*digests*) permettent à un cache d'obtenir une vision globale du contenu des autres caches avec lesquels il coopère. L'objectif est de limiter le surcoût en communications réseau que les autres protocoles de coopération induisent. Pour ce faire, chaque cache établit dans une représentation condensée[3] une liste des documents qu'il possède, et la communique périodiquement aux autres caches du système.

Lorsqu'un client demande un document, le cache commence par vérifier s'il en possède une copie. Dans la négative, il vérifie si un autre cache en possède une copie en consultant les résumés de contenus des autres caches, et donc sans avoir besoin d'effectuer une communication supplémentaire. Il permet donc d'économiser la bande passante dédiée à la communication entre caches, ainsi que d'accélérer la recherche des documents au sein d'un système de caches coopérant.

En utilisant ce procédé, Fan *et al.* [FCAB98] ont mesuré une réduction de la bande passante utilisée pour la communication inter-caches de plus de 50%, une réduction de 75 à 95% de la charge CPU dédiée à la communication inter-cache, tout en maintenant les mêmes taux de réussite que le protocole ICP. Rousskov et Wessels

[3]Les listes sont codées sous forme d'un filtre de Bloom, qui consiste essentiellement en un tableau de bits indexés par la valeur du codage MD5 de chaque URL.

[RW98] ont également mis en œuvre ce procédé dans le cache Squid, et obtenu des résultats similaires.

Les deux problèmes principaux sont la taille des résumés, qui bien que condensés peuvent être d'une taille importante (de 200 kilo-octets à 2 méga-octets pour un cache de taille moyenne [Rou98]), ainsi que la possibilité d'obtenir des résultats incorrects. Un document peut être reporté présent et avoir entretemps été supprimé ou bien correspondre à un document donc la clé d'indexation est identique à celle du document demandé. Inversement il peut être reporté manquant et avoir entretemps été conservé par un cache. Un réglage de la fréquence de mise à jour des résumés s'avère donc nécessaire. De plus, il est possible de réduire encore la bande passante utilisée pour la transmission des résumés en n'envoyant pour un résumé que les différences par rapport à l'ancienne version.

1.7.6 Les politiques de cohérence

Les politiques de cohérence sont destinées à s'assurer de la validité des données cachées par rapport aux données originales. Cette validité peut être mise en cause lors de la modification d'un document ou de sa suppression. Il faut donc s'assurer que les copies des documents correspondent à la version originale. On peut distinguer deux types de cohérence : la forte et la faible. Dans un modèle de cohérence forte, on garantit que les copies d'un document correspondent en permanence à la version originale. Le maintien de ce type de cohérence implique des moyens de notification importants, ce qui gêne notamment le passage à l'échelle d'un tel système. Or le WWW est un réseau mondial dont l'étendue est incontrôlable. On utilise donc plutôt dans ce contexte un modèle de cohérence faible ou relâchée, dans lequel on tolère des incohérences entre les copies et le document original.

Les politiques de cohérence dans les caches web assurent généralement une cohérence faible, même si des recherches ont été menées dans le domaine de la cohérence forte [LC97]. Quatre techniques de maintien de la cohérence [DP96] sont utilisées dans les systèmes de cache : la vérification périodique, les notifications d'invalidation, l'utilisation d'une durée de vie et la validation lors des demandes d'accès. Une autre approche passant par l'intégration de la cohérence à la politique de remplacement est utilisée par le projet Relais [MB97], que l'on présente dans le paragraphe 1.7.7.

La vérification périodique consiste pour le cache à comparer périodiquement la copie de l'objet avec la version originale. En cas d'incohérence, la copie obsolète est supprimée et une nouvelle copie à jour est effectuée. On peut trouver cette technique mise en œuvre dans les caches locaux des navigateurs web. Le navigateur *netscape* permet par exemple de spécifier si la validation du document caché doit être effectuée lors de chaque accès, une seule fois par session ou bien jamais.

De manière symétrique, au lieu de faire vérifier la cohérence par les caches, on peut mettre en œuvre un mécanisme de notification d'invalidation au niveau des serveurs web. Chaque serveur doit alors garder une trace des caches possédant une

copie de ses documents, et notifier chacun des caches en cas de modification ou de suppression. Cette méthode, qui pourrait fournir une bonne cohérence, est cependant clairement peu généralisable, à cause des problèmes de passage à l'échelle qu'elle induit. Une variante intéressante de ce mécanisme est l'approche de Krishnamurthy et Wills [KW98], qui proposent de profiter des envois de documents par les serveurs pour ajouter, en utilisant une technique de *piggybacking*, les informations permettant d'invalider les documents du serveur que pourrait posséder le cache.

Une technique plus utilisée repose sur une estimation de la durée de vie (*Time To Live*) des documents [SG96]. Cette durée de vie peut être fournie par le serveur d'origine, sous la forme d'une date d'expiration. Mais rares sont les serveurs proposant cette information [Gla94, WM99]. Il est donc nécessaire de recourir à diverses estimations, utilisant principalement des heuristiques [FGM+97]. Lorsque la durée de vie est écoulée, le document est considéré comme invalide, et est donc susceptible d'être supprimé du cache. La copie ne peut pas être renvoyée au client sans validation préalable.

Cette technique peut être utilisée en conjonction avec la validation lors des demandes, qui utilise un type de requête HTTP particulier appelé `If-Modified-Since` [FGM+97]. Avec ce mécanisme, le cache effectue lors de chaque demande de document une requête vers le serveur original, en précisant au serveur qu'il ne faut transmettre à nouveau le document que si ce dernier a été modifié depuis une certaine date, correspondant à la date de chargement de la copie.

1.7.7 Les politiques de remplacement

La politique de remplacement joue un rôle important au sein d'un cache. Elle détermine les documents devant être supprimés du cache afin de libérer de la place pour de nouveaux documents. À ce titre, les algorithmes de remplacement tentent de maximiser le taux de réussite en essayant de préserver les données les plus susceptibles d'être à nouveau demandées, suivant divers critères. De nombreux algorithmes ont été proposés. Il est possible de les classifier en trois catégories [Wan99, AWY99] : les politiques traditionnelles, les politiques utilisant des clés et les politiques utilisant des fonctions de coût. Nous allons en voir quelques exemples.

Politiques traditionnelles

Le principe le plus souvent utilisé pour déterminer la probabilité d'accès futur à un document est l'extrapolation à partir de l'historique des accès précédents. Les deux algorithmes les plus simples sont *LRU* (*Least Recently Used*) et *LFU* (*Least Frequently Used*), prenant en compte respectivement le dernier temps d'accès et la fréquence d'accès à chaque document.

LRU *Least Recently Used* Cet algorithme est très simple à mettre en œuvre, et est utilisé dans de nombreux caches, autres que des caches web. Il supprime en

priorité les objets qui n'ont pas été utilisés depuis longtemps.

Dans la mise en œuvre que l'on peut trouver dans Squid [Wes98b], une tâche de fond supprime périodiquement (par défaut, toutes les secondes) des objets, en se basant sur leur position dans une liste triée par date d'accès.

Pour sélectionner les objets à supprimer, Squid détermine parmi un ensemble d'objets lesquels peuvent être supprimés, sur la base d'un certain nombre de facteurs. Si l'objet est en train d'être servi à un client, ou bien chargé depuis un site original, il n'est pas supprimé. Si la date du dernier accès est supérieure à un âge limite, l'objet est supprimé.

L'âge limite est calculé dynamiquement à partir de la taille du cache et de deux bornes d'occupation[4]. Lorsque l'occupation du cache est proche de la borne inférieure, l'âge limite est élevé afin de limiter le nombre d'objets supprimés. Lorsque le taux d'occupation approche de la borne supérieure, l'âge limite diminue, ce qui entraîne une augmentation du nombre d'objets supprimés. Cet âge limite évolue de manière exponentielle en fonction du taux d'activité du cache. En effet, en régime de fonctionnement, il représente le temps nécessaire pour remplir le cache si le taux d'accès reste constant. Il varie généralement entre 1 et 10 jours.

Le module de cache de la plate-forme web Jigsaw [W3Ce] utilise également ce mécanisme.

LFU *Least Frequently Used* [WA97] Cet algorithme prend en compte les fréquences d'accès. Pour chaque accès à un objet, son compteur de référence est incrémenté. On supprime alors de préférence les documents les moins populaires, i.e. ceux dont le compteur a la plus petite valeur. Outre le fait que sa complexité algorithmique est plus élevée que celle de *LRU*, cet algorithme peut notamment souffrir d'un problème de pollution [DAP99] : si un objet populaire (très fréquemment demandé) devient impopulaire, il reste toutefois longtemps dans le cache.

Politiques à clés

Le seul critère de taux de réussite n'est pas représentatif à lui seul des performances, objectives ou perçues, d'un cache. Si l'on désire optimiser selon d'autres critères tels la bande passante économisée, les dates d'expiration ou les temps d'accès, il faut non seulement prendre en compte les fréquences d'accès aux documents, mais également leurs tailles respectives ou les délais de chargement des documents. Ainsi, d'autres algorithmes plus élaborés ont été proposés, parmi lesquels on peut distinguer ceux utilisant des clés [AWY99]. Ces algorithmes effectuent un premier tri basé sur une clé, puis départagent les éventuelles égalités en utilisant une deuxième clé, etc.

[4]Par défaut, la valeur d'occupation basse est de 90% et celle d'occupation maximale est de 95% de la taille totale de l'espace de stockage.

Size Cet algorithme [WAS+96] supprime les objets dont la taille est la plus importante.

Lowest Latency First Cet algorithme [WA97] utilise les temps de chargement comme critère, et supprime de préférence les documents les plus rapidement accessibles.

LRU-MIN C'est une variante de *LRU*, ajoutant un biais en faveur des objets de petite taille. S'il existe dans le cache des objets d'une taille supérieure à une valeur S, *LRU-MIN* [ASA+96] supprime parmi ces objets celui utilisé le moins récemment. S'il n'existe plus d'objets de taille supérieure à S, on répéte le mécanisme avec une taille de $S/2$.

Hyper-G Cet algorithme [WAS+96] est un raffinement de *LFU*, utilisant successivement comme discriminants les fréquences d'accès, puis les temps de dernier accès et enfin les tailles des documents.

Politiques à fonctions de coût

Une dernière catégorie d'algorithmes cherche à raffiner la sélection de documents à supprimer en utilisant une fonction de coût paramétrée par différents critères tels que le dernier temps d'accès, la date d'entrée dans le cache, le coût de transfert, la date d'expiration, etc.

GreedyDualSize L'algorithme *GDS* [CI97] associe un coût à chaque objet. Les auteurs proposent plusieurs variantes, prenant en compte entre autres la taille du document, son temps de chargement, ... Ils se sont inspirés d'algorithmes développés dans le cadre de caches de données de taille uniforme à coût d'accès variable. *GreedyDual* [You94] est en fait un ensemble d'algorithmes incluant une généralisation de *LRU* et une généralisation de *FIFO*. Les auteurs de *GreedyDualSize* ont étendu cet ensemble d'algorithmes au cas de données de tailles et de coûts variables.

Hybrid L'algorithme *Hybrid* [WA97] associe une fonction d'évaluation de l'utilité d'un document et supprime les documents ayant les plus petites valeurs d'utilité. Les paramètres pris en compte pour évaluer cette utilité sont le temps de chargement, le nombre de références et la taille de chaque document. C'est une variante de la politique *Bolot/Hoschka* [BH96], regroupant les valeurs mesurées par serveur plutôt que par document, afin de simplifier le stockage des informations, et prenant en compte les nombres d'accès – comme *LFU* – plutôt que la date de dernier accès.

Lowest Relative Value L'algorithme *LRV* [Riz98] calcule un indice de valeur relative des documents, prenant en compte notamment la taille, le coût de chargement ainsi que la probabilité d'accès au document.

Remarques

De nombreux algorithmes sont principalement axés sur l'optimisation d'un critère particulier, ce qui en rend les comparaisons difficiles. Par exemple, *Size* est axé sur l'occupation du cache, *LFU* sur la conservation des documents les plus populaires, etc. Williams *et al.* [WAS+96] soulignent le fait que les politiques prenant en compte le facteur taille sont les plus efficaces.

De plus, les différentes études de comparaison utilisent généralement une simulation reposant sur des traces. La performance des algorithmes de remplacement dépendant en grande partie des caractéristiques des accès, et les traces ne représentant qu'un ensemble restreint de catégories d'utilisation, les mesures de qualité ne peuvent être exhaustives. Il faudrait les raffiner avec les études des différentes classes de comportements d'utilisateurs [ALF98, AFA97, YJGMD96].

En outre, les différents algorithmes décrits ci-dessus ont des exigences très disparates en termes de complexité spatiale et temporelle. Certaines politiques, telles *LRU* ou *Size*, peuvent se contenter d'une simple liste ordonnée et ne prennent en compte qu'un paramètre simple. *Hybrid* nécessite par contre de conserver en plus de la liste des documents des informations sur les délais de chargement depuis chaque serveur, ce qui peut être rédhibitoire, particulièrement en termes de besoin de stockage et de mise à jour des informations.

1.7.8 Interaction entre politiques de remplacement et de cohérence

Les politiques de remplacement sont à étudier conjointement avec les politiques de cohérence. En effet, l'objectif des politiques de remplacement est de supprimer les documents les moins importants, et celui des politiques de cohérence est de garantir la validité des informations présentes. On peut donc envisager une application de la politique de cohérence permettant de supprimer les documents obsolètes avant de commencer à appliquer la politque de remplacement. Dans la majorité des études et des implémentations, les deux aspects sont dissociés. Quelques projets seulement [SSV99] ont tenté d'intégrer les deux aspects.

L'influence de la politique de remplacement est surtout perceptible dans le cas de caches de taille réduite, i.e. 15 à 25 % de la taille maximale nécessaire. L'impact de la politique de cohérence est quant à lui plus sensible pour les caches de plus grande taille [KW99b, Pie99].

1.7.9 L'accès aux ressources

Le placement des données peut être considéré au niveau d'un système de caches, prenant en compte les chemins d'accès aux fichiers pour choisir le nœud de stockage du document. Les politiques de coopération intègrent peu cet aspect. Korupolu et Dahlink [PfLSDC99] introduisent la notion de placement coordonné dans un réseau

de caches et proposent deux variantes de placement coopératif. Les auteurs étudient également l'influence de la stratégie de remplacement dans un tel contexte. De plus, à l'intérieur d'un nœud de système de cache, la manière dont sont stockées les données peut avoir un impact sur l'organisation et les performances du cache.

La propriété de localité de référence traduit le fait qu'un document HTML fait couramment référence à de nombreuses images intégrées au document, et qu'en conséquence un accès au document entraîne dans la majorité des cas un accès aux images associées. Les systèmes de stockage classiques ne prennent pas en compte cette propriété dans leur organisation interne. Quelques projets de recherche se sont penchés sur cet aspect. Ainsi, Maltzahn *et al.* [MRG99] et Markatos *et al.* [MKPF99] adoptent des approches similaires, s'appliquant sur un système de fichiers classique, consistant à stocker dans un même fichier différents document de taille réduite ou liés par la propriété de localité. Shiver *et al.* [SGHS01] vont plus loin en développant un nouveau système de fichiers, Hummingbird, adapté aux besoins des caches web. Hummingbird exploite la propriété de localité en gérant des grappes de documents (*cluster*), automatiquement ou grâce à des indications du cache. De plus, comme dans le cache web Cheetah [KEWG96], les documents sont considérés comme des unités immuables, ce qui simplifie leur gestion.

Outre l'intégration de la propriété de localité, d'autres aspects sont liés à la manière de placer les données. Ainsi, le partitionnement de l'espace de stockage [Pie99] peut introduire des améliorations de performance. Duarte *et al.* [MAM98] partitionnent l'espace de stockage selon la taille des documents à stocker, et montrent que cette méthode permet d'obtenir d'excellents résultats suivant les deux métriques de taux de réussite et de taux de réussite pondéré par la taille (voir le paragraphe 1.8.2). Arlitt *et al.* [ACD+98] introduisent la notion de caches virtuels cascadés permettant d'appliquer plusieurs politiques de remplacement. L'espace de stockage est partitionné en plusieurs caches virtuels disposant chacun d'une stratégie de remplacement différente. Les documents (ou plutôt leurs références) sont initialement maintenus dans le premier cache virtuel. Lors de l'application de la politique de remplacement, les documents ne sont pas supprimés mais migrent vers le cache virtuel suivant. Seul le dernier cache virtuel supprime effectivement ses éléments. Les différentes politiques de remplacement mettant l'accent sur des métriques différentes, on obtient alors une amélioration des performances selon une combinaison des différentes métriques.

Enfin, il est possible de prendre en compte des facteurs de qualité au niveau du stockage des données. Ortega *et al.* [OCAV97] proposent des stratégies spécifiques pour les données de type image. En fonction de l'espace de stockage disponible et des vitesses de transfert, leurs stratégies peuvent stocker une version dégradée de l'image afin d'en accélérer l'accès.

1.8 La mesure des performances

1.8.1 Objectif des mesures de performances

L'objectif des mesures de performance est double. Dans un contexte de mise en place d'un service de cache web, on veut pouvoir évaluer les performances futures du système envisagé qui peut varier dans son dimensionnement, dans son architecture, dans sa configuration ou dans les outils mis en œuvre (typiquement le choix entre divers produits disponibles).

Dans un contexte de cache web déjà en place, il s'agit de s'assurer de l'adéquation des performances par rapport aux évaluations précédemment effectuées, et plus généralement du fonctionnement correct du système. Les mesures peuvent alors être utilisées pour adapter certains paramètres du système.

1.8.2 Des critères multiples

La qualité des caches web est généralement évaluée selon trois critères [BO00] : la vitesse, la capacité de passage à l'échelle (*scalability*) et la robustesse, auxquels on peut ajouter la quantité de bande-passante économisée, la réduction de charge sur les serveurs d'origine.

L'utilisateur final est plus directement concerné par des critères temporels, principalement la réduction des latences (temps de réaction) et la diminution des temps de chargement. La fraîcheur des documents proposés entre également dans la perception globale de la qualité du service.

Le fournisseur d'un service de cache a pour objectif de satisfaire ses utilisateurs, et prend donc en compte les mêmes critères. Cependant, son intérêt est également de réduire au maximum l'encombrement de ses réseaux. Il s'intéresse donc à la quantité de bande passante économisée. Dans une optique de déploiement du système, les performances en termes de débit soutenable (en requêtes par seconde ou en *hits* par seconde), de capacité de redimensionnement ou de tolérance aux pannes sont également fondamentaux.

Les taux de réussite

Le critère d'évaluation le plus courant pour les caches informatiques pris dans leur ensemble (cache mémoire, cache d'instructions, etc.) est généralement le taux de réussite (*Hit Ratio*), qui conditionne plus ou moins directement l'amélioration des performances. On peut le définir ainsi [SSV99] :

$$HR = \frac{\sum_i h_i}{\sum_i f_i}$$

où h_i est le nombre de références au document i qui ont été satisfaites par le cache, et f_i est le nombre total de références au document i.

Si l'on suppose que l'on est en présence d'un cache de capacité illimitée, on peut mesurer ainsi le gain maximum que l'on peut espérer atteindre. Les études les plus répandues [Cor98, ASA+96, WMS98], basées sur des traces diverses, indiquent que la limite maximale en termes de *taux de réussite* est de 30 à 50%.

Cependant, le cas des caches web est un peu différent de celui des caches mémoire, comme mentionné dans la partie 1.7.1. Une particularité notable est la non-uniformité des tailles des éléments cachés. Dans un cache mémoire, les lignes de caches sont de même dimension. Cette distinction apparaît importante au regard d'un des objectifs des caches web qui est de limiter la quantité de données transférées sur un réseau. On introduit donc un nouveau critère, le *Byte Hit-Ratio* [SSV99], qui est le taux de réussite pondéré par la taille de chacun des documents. On obtient donc une mesure du nombre d'octets économisés.

$$BHR = \frac{\sum_i h_i.s_i}{\sum_i f_i.s_i}$$

où s_i représente la taille du document i.

Les critères temporels

On peut également vouloir prendre en compte les vitesses de chargement des documents. Dans l'hypothèse d'un débit constant, la vitesse de chargement est proportionnelle à la taille. Cependant, cette hypothèse n'est clairement pas vérifiée dans le cas général. Il faut donc introduire une notion de délai de chargement relatif à chaque document, ainsi que de délai de validation si l'on prend en compte l'impact de la politique de cohérence, et on peut alors définir le taux de réduction de délai, ou *Delay Savings Ratio*, comme le propose [SSV99] :

$$DSR = \frac{\sum_i (h_i.d_i - v_i.c_i)}{\sum_i f_i.d_i}$$

où d_i représente le délai moyen pour accéder à un document depuis le cache, v_i représente le nombre de validations effectuées pour le document i, et c_i représente le délai moyen de validation d'un document.

Cependant, comme le soulignent Caceres *et al.* [CDF+98], ces paramètres ne sont pas représentatifs de la qualité de service perçue par l'utilisateur. Ce dernier est plus sensible à la vitesse de réaction du système, i.e. au temps écoulé entre l'activation d'un lien hypertexte et le début de l'affichage ou du chargement du document demandé. Caceres *et al.* [CDF+98] proposent une simulation de bas niveau, afin de prendre en compte notamment les temps d'établissement de connexion dûs à TCP. Le mécanisme de connexion persistante introduit dans `HTTP1.1`, ainsi que les

techniques de préchargement mentionnées dans la partie 1.7.3 apportent une grande amélioration sur cet aspect.

Dans [ASF98], Afonso *et al.* proposent une nouvelle métrique pour mesurer la qualité de service fournie par un cache : le temps de réponse à une requête. Les dernières versions de PolyGraph [RW00] proposent également de mesurer ce paramètre en fonction de la charge du cache, ce qui permet de déterminer la limite en nombre d'utilisateurs du système.

Les métriques d'exploitation

Dans une optique de passage à l'échelle, il est intéressant de savoir si un cache est susceptible de supporter une augmentation de charge. Ainsi, on peut définir d'autres critères [DAP99, RS99] tels que la charge du processeur, la charge des entrées/sorties, les taux d'utilisation du disque, etc. On retrouve notamment ce genre d'informations dans la plate-forme d'évaluation de performances PolyGraph [RW00].

Toujours dans la perspective d'augmentation de charge, les méthodes d'évaluation de caches incluent souvent une évaluation du comportement lors d'une surcharge en nombre de requêtes [AC98, RW00], ainsi que la vitesse de réaction à un arrêt du système [RW00]. Certains produits de caches comme celui d'Infolibria [RW99] intègrent en effet un mécanisme de court-circuit du cache en cas de surcharge. Les requêtes sont alors transmises directement, assurant ainsi un accès permanent au web même si les performances sont alors dégradées.

On note sur ce dernier exemple l'apparition d'un mécanisme d'adaptation dynamique de comportement, mis en œuvre en partie de manière matérielle (le mécanisme de court-circuit du cache). Basé sur l'observation de la charge du cache, le système choisit d'activer le comportement normal (conservation de copies) ou un comportement dégradé (transfert direct des données, sans conservation de copie).

Pour un fournisseur de service de cache, outre la satisfaction des utilisateurs, la mesure des économies réalisées peut constituer une métrique pertinente. Ces économies peuvent relever d'économies purement techniques, telles que l'optimisation de l'utilisation des ressources réseau, ou être traduites ensuite en termes financiers [dJ97, WMS98], prenant en compte à la fois les économies réalisées grâce au système et les frais d'installation et de maintenance du système.

Les critères plus subjectifs

Outre les temps de latence, qui jouent directement sur la perception de l'utilisateur, le critère de fraîcheur des documents s'avère intéressant. Warfield *et al.* [WMS98] soulignent néanmoins la difficulté de mesurer ce critère de fraîcheur des documents. On pourait effectivement le formuler comme le taux de documents périmés délivrés par le cache. Mais savoir si un document est périmé revient à être capable de connaître la version à jour du même document, ce qui n'apparaît pas

dans les traces de simulation.

Kelly, Chan et Jamin [KCJ99] posent l'hypothèse que les bénéfices apportés par les caches (typiquement la réduction dans les latences) sont perçus différemment par les divers utilisateurs. Ils proposent donc d'intégrer ce coefficient dans les métriques de qualité du système de cache.

Des métriques spécifiques

Il est possible pour certains mécanismes particuliers de développer des métriques spécifiques adaptées au mécanisme. Par exemple, dans le cas d'une mise en œuvre de préchargement [FCLJ99], on peut vouloir introduire la notion de nombre de requêtes satisfaites par le cache grâce au mécanisme de préchargement, ainsi que la bande passante inutilement utilisée pour précharger des documents non demandés par la suite.

1.8.3 Classification des méthodes de mesure

Davison [Dav99b] propose une classification des méthodes d'évaluation de caches suivant la source du trafic utilisée et l'implémentation de l'algorithme de mesure.

Source du trafic

On peut considérer principalement quatre sources de trafic : les traces générées artificiellement, l'analyse des fichiers d'historique des caches, les captures de requêtes (*traces*) et l'utilisation de requêtes réelles. Chaque méthode apporte différentes propriétés, dans les domaines du réalisme ou encore de la reproductibilité des mesures.

Traces artificielles Les traces générées sont souvent utilisées pour produire des environnements de test inexistants afin de répondre à des questions telles que l'impact d'un doublement du nombre de requêtes. Il est également plus simple de simuler des environnements particuliers en termes de profil de clients ou de répartition de documents. Cette simulation est indispensable par exemple dans les études sur les évolutions des besoins dans divers scénarios comme l'accroissement du nombre d'utilisateurs, l'évolution du comportement des utilisateurs ou encore l'évolution des types de données et de leurs caractéristiques.

Il a été établi [Gla94, RW00, BCF+99] que la distribution des requêtes sur des objets web suit généralement une distribution *Zipf-like*. Dans une telle distribution, la probabilité relative que le i-ème document le plus populaire soit demandé est proportionnel à $1/i^{\alpha}$, avec $\alpha < 1$. Le paramètre α varie suivant le profil de la population étudiée. Ainsi, pour valider ce modèle, ce paramètre a été calculé pour différents

types de traces [BCF+99]. La valeur de α était comprise, suivant la trace, entre 0.64 et 0.83.

L'outil de mesure de performances de caches web *PolyGraph* [RW00] permet d'utiliser cette distribution, ou bien une distribution uniforme, lors de la génération des traces. D'autres paramètres tels que la localité temporelle ou le taux de cachabilité peuvent affiner la description de la distribution simulée.

Analyse des fichiers de données Des outils permettant d'analyser les performances du cache au vu de ses fichiers d'historique (*fichiers de log*) ont été développés. L'avantage de cette approche est sa simplicité de mise en œuvre. En effet, les caches web conservent de toute façon dans leur configuration par défaut une trace des requêtes qu'ils ont traitées. Il suffit donc de les analyser pour extraire des informations, sans avoir à mettre en place de systèmes de mesure supplémentaires.

`calamaris` est un outil permettant d'analyser les fichiers d'historique de nombreux produits de caches web, dont Squid [Wes98b], NetCache [Dan98], Inktomi Traffic Server [Ink], Netscape/iplanet Web Proxy Server [Net], etc. Il produit des statistiques synthétiques du trafic observé, comprenant notamment les moments de pic de trafic pour chaque protocole, ainsi que les taux de réussite classés par type d'objet.

`squeezer` est un outil développé spécifiquement pour analyser les historiques des caches Squid. Il faut noter que le développeur de cet outil a précisé dans sa documentation un guide sur l'interprétation des résultats. Ces derniers incluent par exemple le taux d'efficacité de la coopération avec des caches voisins ainsi que le taux de réussite suivant le type d'objet.

Connaître le taux de réussite suivant le type d'objet considéré permet de régler plus finement les paramètres de rafraîchissement des objets (*refresh patterns*) [Not99].

Cependant, malgré sa facilité de mise en œuvre, cette méthode présente plusieurs inconvénients liés en grande partie à son imprécision. En effet, comme le décrit Brian Davison [Dav99c], les fichiers d'historique ne présentent qu'une vision partielle du trafic. Ils ne fournissent généralement pas d'information sur la validité des documents, ou sur certains entêtes utiles comme les directives de contrôle de cache. Enfin, les informations temporelles sont souvent incomplètes et ne mentionnent que rarement les latences ou les temps d'accès détaillés.

Traces capturées Capturer des requêtes réelles permet d'obtenir une base de simulation plus réaliste. De plus, elles offrent la possibilité de rejouer plusieurs fois le même jeu de données, permettant ainsi de comparer différentes configurations sur une base commune. On trouve ainsi un certain nombre de *traces standard*, commes les traces DEC [KMM97], utilisées comme base de comparaison dans de nombreuses études. Cependant, leur durée de validité est sujette à caution du fait de la courte durée de vie d'un document sur le WWW [DFKM97], ce qui peut entraîner des

incohérences lors des simulations.

De plus, les traces peuvent être plus ou moins exactes suivant la méthode utilisée pour les récolter, notamment à l'égard de l'ordonnancement des requêtes et des informations plus élaborées sur les documents demandés comme la validité des documents servis.

C'est néanmoins le moyen le plus utilisé actuellement pour l'évaluation et la comparaison des performances de caches. Les traces peuvent être prélevées au niveau du cache en utilisant ses fichiers d'historique ou en instrumentant le cache [RS99] pour obtenir des données plus précises, au niveau du client en procédant à une instrumentation du logiciel de navigation utilisé [CBC95] ou encore au niveau intermédiaire en capturant le trafic [PM00].

Requêtes réelles Cette méthode consiste à intercepter un flux réel de requêtes au fur et à mesure de leur passage sur le réseau, et à les soumettre à plusieurs caches en parallèle. Les mesures basées sur des requêtes réelles posent le problème de la non-reproductibilité. Cette méthode est donc généralement utilisée pour obtenir des mesures sur une configuration particulière, et non pour effectuer des comparaisons entre différentes configurations.

Ainsi, Pandora [PM00] est un système flexible de supervision de réseau. Une configuration en a été proposée pour la supervision plus spécifique de caches web distribués. Une des applications consiste, dans un réseau utilisant des caches, en l'extraction de traces d'accès web prenant en compte les biais introduits par les caches. Une autre application permet de mesurer une qualité de mesure instantanée d'un système de caches répartis, en fonction de critères personnalisables.

Mise en œuvre de l'algorithme de mesure

La mesure des performances d'un système de caches peut être effectuée de trois manières : en réalisant une simulation complète du système de cache et du réseau qu'il utilise, en mesurant les performances d'un cache réel sur un réseau isolé dont on maîtrise les paramètres, ou bien en effectuant les mesures sur un système réel sur un réseau en production.

Simulation complète La simulation totale du système de cache et du réseau représente la solution la plus simple à mettre en œuvre. Cette simplicité s'acquiert au détriment du réalisme des mesures, le modèle utilisé n'étant qu'une approximation de la réalité, tant au niveau du cache (tous les aspects ne sont pas pris en compte, notamment ceux de plus bas niveau) que du réseau simulé (on ne simule que rarement les temps de chargement des documents).

Saperlipopette ! [Pie99] est un simulateur de caches web distribués, développé dans le cadre de la thèse de Guillaume Pierre au sein de l'INRIA. Il peut être utilisé

pour évaluer *a priori* la qualité de service qu'il est possible d'obtenir dans chaque configuration potentielle de l'infrastructure de caches distribués, tant en termes de dimensionnement que de placement.

Saperlipopette ! effectue une simulation à base d'événements discrets, fonctionnant en deux phases. Dans un premier temps, une capture du trafic réseau dédié au web est effectué sur le réseau destiné à utiliser le cache. Cette phase d'audit, d'une durée de quelques mois, permet de cerner les motifs récurrents d'accès à des documents, l'historique des accès ainsi que les caractéristiques du réseau. Dans un deuxième temps, les données capturées sont utilisées comme base de simulation, permettant ainsi de tester chacune des configurations envisagées.

Cache réel sur un réseau isolé Il est possible de limiter les variations dues à une charge réseau fluctuante en effectuant des mesures sur un cache réel fonctionnant sur un réseau isolé, dont le trafic est maîtrisé. Cette méthode, outre le fait qu'elle produit généralement des mesures surévaluées, permet difficilement de prendre en compte des facteurs tels que les délais d'acheminement des documents, la charge des réseaux, etc.

Ainsi, la plate-forme d'évaluation de caches PolyGraph [RW00] intègre un module de génération de trafic, simulant de bout en bout les requêtes des clients et les réponses des serveurs, allant jusqu'à générer des contenus aléatoires afin de tester certaines fonctionnalités des caches (comme la restriction d'accès basée sur le contenu des documents).

Cache réel sur un réseau réel Les mesures effectuées sur un cache réel fonctionnant sur un réseau réel fournissent le plus fort degré de réalisme, mais ne permettent pas de reproduire les mesures de manière cohérente. De plus, on ne maîtrise alors plus l'environnement de mesure de bout en bout, et on ne peut notamment plus appliquer de contraintes particulières aux serveurs d'origine.

Davison [Dav99a] propose par exemple une architecture d'évaluation simultanée de caches (*Simultaneous Proxy Evaluation*), permettant de dupliquer les requêtes observées sur un réseau et de les envoyer sur plusieurs caches différents. Elle permet donc une comparaison plus réaliste des performances de différents caches.

1.8.4 Un environnement dynamique

L'environnement d'exécution des systèmes de caches web est particulièrement dynamique, à plus d'un titre. En effet, un système de cache est par définition intégré dans un réseau informatique ouvert, utilisé conjointement par de nombreux utilisateurs et pour diverses applications. Le taux d'utilisation de la ressource réseau, que l'on peut mesurer par exemple au niveau du lien à Internet d'une institution [MRGM99] ou du *backbone* d'un opérateur [TMW97], est donc très variable. De plus,

suivant la position du cache dans une hiérarchie, les conditions de fonctionnement peuvent être très différentes.

De plus, les ressources internes du système sont également variables. On peut par exemple mentionner les ressources disques, qui sont amenées à évoluer au cours du temps et de l'utilisation du cache [RS99], ou encore les ressources des interfaces réseau, qui comprennent non seulement la liaison entre l'institution faisant fonctionner le cache et Internet, mais également toutes les connexions vers les utilisateurs.

Enfin, les modalités d'utilisation du cache sont également particulièrement versatiles. D'une manière globale, la charge d'un cache, mesurée en nombre de requêtes traitées par seconde, peut être représentée par une courbe en cloche, avec un maximum atteint en journée et un minimum la nuit. Cependant, Rousskov et Soloviev [RS99] ont mené une étude sur une hiérarchie de caches, et ont été amenés à distinguer la charge d'un cache de bas niveau, institutionnel par exemple, de celle d'un cache de haut niveau. Ces derniers ne profitent en effet pas forcément d'une répartition temporelle des requêtes, et présentent peu de variations sur une période de 24 heures.

Dans le cas général, des périodes de sous-utilisation du réseau peuvent être identifiées. Maltzahn *et al.* [MRGM99] proposent par exemple d'en tirer parti, afin de précharger les données qui seront probablement demandées lors de la période d'utilisation intensive suivante.

En descendant plus dans le détail des utilisations du serveur, on peut distinguer deux motifs de variation principaux : au niveau des utilisateurs, et au niveau du type des données manipulées.

Il est possible d'identifier différentes classes d'utilisateurs [ALF98, AFA97, YJGMD96, AWA+95], suivant leurs comportements. Williams *et al.* [WAS+96] étudient ainsi les performances respectives des politiques de remplacement en se basant sur quatre catégories d'utilisateurs présentant des motifs d'utilisation différents. De même, Kelly, Chan et Jamin [KCJ99] postulent l'existence de plusieurs catégories d'utilisateurs aux besoins divers en termes de cache. Ces différentes classes d'utilisateurs peuvent utiliser un cache au même moment, ou bien s'en partager le service sur des périodes déterminées et relativement distinctes.

En ce qui concerne les données manipulées, elles peuvent présenter des caractéristiques diverses au regard de la durée de validité, des besoins en stockage, de la fréquence d'accès, etc. Ainsi, Williams *et al.* [WAS+96] proposent de partitionner l'espace disque disponible afin de dédier certaines zones à des types de données particuliers, ce qui revient à appliquer une stratégie de remplacement distincte suivant le type de données.

Pierre *et al.* [PKvST00] adoptent une stratégie aux idées similaires, même si la mise en œuvre diffère quelque peu. Les auteurs, ayant identifié les besoins variés de documents différents en termes de réplication, proposent d'associer à chaque document une politique de réplication spécifique, en argumentant notamment qu'une

politique unique ne peut satisfaire tous les cas, et qu'il est donc nécessaire d'avoir recours simultanément à plusieurs politiques spécialisées.

Le cache Squid [Wes98b], ainsi que les caches commerciaux NetCache [Dan98] et CacheEngine [Cis], permettent d'ailleurs dans leurs options de configuration de spécifier pour chaque type de document les paramètres d'estimation de validité. Nottingham [Not99] a proposé sur cette base un outil d'analyse permettant d'assister l'administrateur de cache dans la détermination de ces paramètres. Une extension mentionnée est l'intégration de l'outil dans le cache, afin d'ajuster automatiquement les paramètres au caractéristiques du trafic observé.

1.9 Les domaines d'amélioration des caches web

Le travail autour des caches web est intense, tant au niveau industriel qu'au niveau académique, les deux niveaux tendant particulièrement à se confondre dans ce domaine. Nous soulignons ici quelques pistes de recherche nous semblant particulièrement intéressantes. Leur point commun est d'introduire un aspect dynamique dans le fonctionnement des caches, afin d'adapter le plus possible le comportement des caches à leurs conditions d'exécution.

1.9.1 Les politiques de coopération

> « *[la grande diversité dans les taux de réussite] suggère qu'un mécanisme de gestion des caches capables d'activer et de désactiver la coopération sera utile.* »[5] [KS98]

Nous avons vu dans le paragraphe 1.7.5 qu'il existe de nombreux protocoles permettant la coopération entre plusieurs caches.

Comme le notent Krishnan et Sugla [KS98], les gains de performances découlant de la coopération des caches proviennent particulièrement de l'augmentation de la taille effective du système de cache, ainsi que du préchargement potentiel effectué par les autres utilisateurs, d'autant plus nombreux que la taille du système de cache est importante.

Cependant, Krishnan et Sugla soulignent également le fait qu'une coopération systématique n'offre pas des performances optimales, notamment à cause de la surcharge réseau qu'entraînent les mécanismes de coopération. Ils proposent donc de définir des métriques permettant de mesurer leur efficacité, telles que la mesure du trafic utile induit par la coopération – qui varie de 2% à 7.5% suivant la politique de coopération utilisée – ou l'augmentation de la charge du cache coopérant – qui varie de 10 à 300% suivant la taille du cache et la politique de coopération.

[5] [the large variability in hit rates] suggests that a dynamic mechanism for managing the proxies and turning cooperation on and off will be useful

Ces métriques doivent permettre de fournir les bases de caches web adaptant les mécanismes de coopération aux conditions d'exécution.

1.9.2 Les politiques de remplacement

> « *Il pourrait être intéressant d'étudier des algorithmes de remplacement adaptatifs.* »[6] [WA97]

Comme présenté dans la partie 1.7.7, de nombreuses politiques de remplacement sont disponibles. Les travaux en cours cherchent à déterminer une politique générique la plus efficace possible.

Quelques travaux [WA97, Mar96] ont émis l'idée, au vu des performances variables des différents algorithmes en fonction du type de trafic, d'adapter dynamiquement ces politiques aux conditions d'exécution. Ainsi, Markatos [Mar96] propose une politique adaptative de remplacement, consistant en une politique LRU paramétrée. Les paramètres sont modifiés aux cours de la vie du système, afin d'optimiser le taux de réussite.

Cependant, il n'existe pas encore de travaux étudiant la possibilité d'adapter dynamiquement l'algorithme lui-même plutôt que ses paramètres dans un cache web.

1.9.3 Modularisation

En ce qui concerne la mise en œuvre et l'évolutivité du code, on constate une tendance à la modularisation. On peut le voir sur deux exemples : Jigsaw et Squid.

Squid a eu un développement progressif, partant d'un code pré-existant et plutôt spécialisé au départ. Depuis peu, des projets de développements ont vu le jour, dans une optique de modularisation de la couche de stockage des informations (`modio` [Cha01]), de la couche de communication (`comm-select`) ou encore de la politique de remplacement. Ces travaux sont cependant effectués sur une infrastructure déjà existante, et leur portée peut parfois être limitée par des contraintes intrinsèques au système existant.

Jigsaw, projet plus récent de plate-forme de développement de services web élaboré par le consortium W3, a dès le départ été conçu de manière modulaire, avec un fonctionnement à base de ressources et de filtres. L'architecture en est présentée dans le chapitre 5. Cette modularité a un coût en termes de performances, mais il semble d'après les développeurs suffisamment faible pour justifier le modèle proposé. Nous avons donc choisi d'utiliser cette base pour évaluer la pertinence de l'autoadaptativité appliquée aux caches web.

[6] [...] adaptive cache replacement algorithms may be worth exploring.

1.10 Conclusion

> « *Pour un système à grande échelle, la plupart des décisions de réplication et de cachage doivent être totalement automatiques, bien que des décisions locales puissent être imposées sur une telle infrastructure.* »[7] W3C [W3Cg]

Dans le paragraphe 1.8.4, nous avons présenté la grande variété de conditions de fonctionnement des systèmes de caches. Cette variété semble naturellement requérir un fonctionnement également dynamique. Cependant, même si le besoin d'adaptation a été identifié [W3Cf, WA97, Mar96, KS98], peu de projets ont vu le jour sur ce sujet.

Jaws [HS99] est une plate-forme permettant de concevoir des serveurs web flexibles et adaptatifs, proposant une approche très modulaire. On y trouve notamment une politique de gestion du parallélisme de connexion ou des entrées-sorties. Ce projet est dédié à l'élaboration de serveurs web, mais ses principes et outils pourraient être utilisés dans le cadre des caches également.

Le projet CacheL [BP99] propose d'intégrer aux caches un langage dédié permettant de spécifier diverses politiques au niveau du cache, comme par exemple les politiques de cohérence, de remplacement ou de routage des données. L'infrastructure proposée permet aux clients, serveurs et caches de spécifier des politiques associées aux objets transmis. Une implémentation, effectuée sur la base du cache Squid [Wes98b], a montré que ce mécanisme pouvait améliorer les performances du système.

CacheL fournit un support intéressant pour effectuer une adaptation du comportement des caches web. Cependant, le projet est orienté vers la possibilité d'une spécification de politiques par diverses entités (administrateurs de caches, utilisateurs) plutôt que vers une adaptation automatique par le cache basée sur l'introspection.

Il existe diverses métriques permettant d'évaluer les performances des systèmes de caches. Leur mesure peut s'effectuer directement dans le cache, ou nécessiter des dispositifs supplémentaires. Cependant, il est possible d'obtenir des informations en temps réel sur le fonctionnement des caches, ainsi que d'analyser les statistiques afin de tenter de prédire l'occurrence de certains événements significatifs tels que des montées en charge soudaines.

Il serait donc intéressant d'associer plus étroitement les mécanismes de mesure au fonctionnement des systèmes de caches, afin d'en optimiser dynamiquement les performances via l'autoadaptation de certaines de ses politiques [Aub99].

[7] « For a large scale system, most replication and caching decisions must be completely automatic, though local policy decisions may be imposed over such an infrastructure. »

CHAPITRE 2

Adaptativité

Diverses acceptions du terme *adaptativité* et des termes connexes tels que *adaptabilité*, *flexibilité*, *viabilité*, *configurabilité*, etc. ont été proposés. De même, plusieurs méthodes d'évaluation ou de classification de systèmes adaptatifs existent. Dans ce chapitre, nous présenterons ces différents termes et méthodes dans le but d'éclaircir l'état de l'art de l'adaptativité, et de constituer une base solide pour la présentation de notre orientation.

Le principe qui sous-tend l'analyse que nous allons conduire est un principe d'évolution, partant du postulat que tout système informatique est un système adaptable, c'est-à-dire amené à évoluer au cours du temps, tant dans son fonctionnement que dans sa structure et sa configuration. Cette adaptation, comme nous allons le voir par la suite, peut être considérée de différents points de vue et à différentes échelles.

Après avoir défini la terminologie utilisée, nous nous intéressons dans le paragraphe 2.2 à l'objectif principal de l'adaptation qui est la *viabilité*. Cet objectif très général peut être atteint de multiples façons par les systèmes informatiques. Nous allons donc nous attacher dans le paragraphe 2.3 à préciser les propriétés fondamentales des systèmes adaptatifs, et nous explicitons les trois grands types d'adaptation que sont l'adaptation de paramètres, l'adaptation de comportements et l'adaptation d'architecture. Ensuite, nous présentons dans le paragraphe 2.5 une classification en deux axes temporels des types d'adaptation. Sur la base de ces différents critères de classification, nous proposons une dénomination plus explicite des mécanismes d'adaptation, permettant de lever quelque peu les ambiguïtés d'interprétation du terme d'adaptation que l'on peut trouver dans le domaine informatique. À titre d'illustration de cette dénomination, nous verrons comment l'objectif général de viabilité se décline suivant les domaines d'utilisation. Le paragraphe 2.6 s'attache à l'étude de points de vue divers sur l'adaptation, qui introduisent souvent un vocabulaire spécifique et par là des angles d'attaque différents sur un sujet similaire. La dénomination proposée permettra enfin de préciser le domaine de l'étude que nous poursuivrons dans la suite de cette thèse, l'autoadaptation dynamique de comportements.

2.1 Terminologie

Nous utilisons les termes définis par Fenton et Hill [FH92], tels que repris par Tekirnedogan [Tek96]. Selon leur terminologie, un *système* est une entité identifiée par une frontière au sein d'un environnement. Il est constitué d'un assemblage de composants connectés de manière organisée. Cet assemblage organisé a un objectif observable qui est caractérisé par la manière dont il transforme les données venant de l'environnement en données retournées à l'environnement.

La plupart des systèmes sont si grands et complexes que la seule manière de les analyser est de les décomposer en *sous-systèmes*. Un *sous-système* est un système à part entière, qui est contenu dans un autre système. Les sous-systèmes doivent concentrer un aspect spécifique et présenter une forte cohésion. De plus, les sous-systèmes doivent être assemblés de manière suffisamment lâche pour permettre de gérer plus souplement leurs interactions.

L'*environnement* est l'espace dans lequel le système fonctionne. L'*environnement important* du système est la partie de l'environnement qui interagit avec le système. La partie qui n'interagit pas avec le système constitue l'*environnement non-important*. L'environnement est donc constitué de l'union de l'*environnement important* et de l'*environnement non-important*. Néanmoins, le terme environnement sous-entend en général le qualificatif *important*.

$$\text{environnement} = \text{environnement important} \cup \text{environnement non-important}$$

La signification des termes d'environnement, de système et de sous-système dépend du niveau d'abstraction considéré. En effet, l'environnement peut être perçu comme un autre système contenant le système étudié. Celui-ci contient à son tour des sous-systèmes, et constitue de fait leur environnement. Enfin, ce raisonnement peut s'appliquer de nouveau aux sous-systèmes. Nous appellerons *domaine* le système étudié et son environnement important :

$$\text{domaine} = \text{système} \cup \text{environnement important}$$

et nous désignerons par le terme *contexte* l'intersection de ces deux éléments :

$$\text{contexte} = \text{système} \cap \text{environnement important}$$

La figure 2.1 représente le diagramme de Venn associé aux définitions ci-dessus.

2.2 Objectifs de l'adaptation

Le besoin d'adaptation découle principalement d'un principe de survie. Ce principe est généralement observé dans les organismes vivants. Le psychobiologiste Hebb

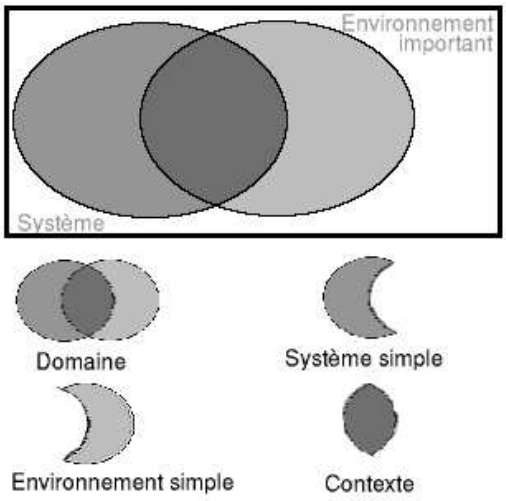

FIG. 2.1 — Modèle utilisé

souligne ainsi que *« le comportement est tout d'abord l'adaptation à l'environnement en fonction des informations sensorielles. Il éloigne l'organisme des situations dangereuses, le rapproche des conditions favorables, ou modifie l'environnement immédiat afin d'augmenter les chances de survie. »*[1]

Ce principe de survie peut également s'appliquer à des systèmes informatiques. En effet, dans ce domaine la survie d'un système informatique est soumise à la condition que le système satisfait un certain nombre de propriétés. Ces propriétés ne sont assurées que si le système s'adapte aux conditions d'utilisation au sens large – qui comprennent donc les besoins des utilisateurs, les ressources disponibles, etc. Dans cette perspective, l'**adaptabilité** (voir paragraphe 2.6.2) est une propriété nécessaire lors de la conception des systèmes informatiques, et conditionne leur **viabilité** (voir paragraphe 2.2.1).

2.2.1 Viabilité

Le concept de **viabilité** est issu des travaux du cybernéticien Stafford Beer, le fondateur de la cybernétique du management[Wie48]. Cherchant à trouver des concepts et des outils permettant d'analyser les systèmes complexes que constituent les grandes organisations humaines, il a proposé un modèle topologique appelé le VSM, pour *Viable Systems Model* [Bee84]. Dans ce modèle, il distingue un ensemble de fonctions qui assurent la **viabilité** de tout système vivant, s'appliquant aux organisations humaines mais également au niveau neurophysiologique des êtres vivants.

[1] « Behavior is primarily adaptation to the environment under sensory guidance. It takes the organism away from harmful events and toward favorable ones, or introduces changes in the immediate environment that make survival more likely. » (Hebb, in *A Textbook of Psychology*)

Le tableau 2.1 précise les correspondances entre les différents domaines considérés (organisation, être vivant, système informatique). Les termes utilisés dans la figure 2.2 se rapportent au domaine de l'entreprise.

Patron de conception	**Organisation**	**Être vivant**	**Système informatique**
Contrôle séparé	Management *vs.* travailleurs	Système nerveux central *vs.* organes	Logique de contrôle *vs.* processus
Contrôle opérationnel	Responsable d'équipe	Contrôle pons-médullaire	Système d'exploitation *vs.* Ressources et applications
Centre régulateur	Agenda de production	Système nerveux sympathique	Ordonnanceur de processus, gestion de la mémoire
Audit sporadique	Comptabilité, Auditeurs externes	Système nerveux parasympathique	Système de monitoring
Contrôle adaptatif	Prévisions, Recherche et développement	Entrées sensorielles du diencéphale, prévisions	Systèmes informatiques autoadaptatifs
Contrôle de supervision	Président, Directeur Général, Président du CA	Cortex, Fonctions cérébrales élaborées	Spécification déclarative de comportement
Alertes	Prix des actions, manifestations, grèves	Douleur, plaisir	Notification de réussite ou d'échec
Composition récursive	Hiérarchies, divisions, équipes	Cellules, organes, organisme	Architecture en couches, assemblage de composants
Boucle homéostatique	Contrats	Pression sanguine, température corporelle	Spécification des interfaces

Tab. *2.1* — Correspondances entre les patrons, les organisations, les êtres vivants et les systèmes informatiques

Le modèle des systèmes viables repose sur une hiérarchie de régulateurs. Un régulateur – que l'on peut également appeler contrôleur – est un composant capable de contrôler un élément du système. L'idée générale est que les régulateurs de haut niveau sont capables de gérer la diversité en filtrant les messages provenant des niveaux plus bas, pour supprimer à la fois les informations qui ne sont pas nécessaires aux objectifs de niveau plus élevé et les informations qui ne sont pas significatives sur le long terme. Les régulateurs de bas niveau sont utilisés pour amplifier la variété

d'actions dont dispose le niveau supérieur. Beer souligne également l'importance de canaux d'informations spécifiques (appelés « canaux algédoniques »), capables de court-circuiter la structure hiérarchique standard.

Une des propriétés essentielles du modèle proposé par Beer est sa faculté de s'appliquer de *manière récursive à différentes échelles*. On peut donc retrouver par exemple à l'intérieur d'un régulateur la même structure complexe que celle du système dont il fait partie, ce qui est illustré dans la figure 2.2 par la présence à une échelle plus réduite de la structure de contrôleurs dans la sphère représentant le processus de production.

Un modèle simplifié, dont les termes se rapportent au monde industriel, en est présenté dans la figure 2.2.

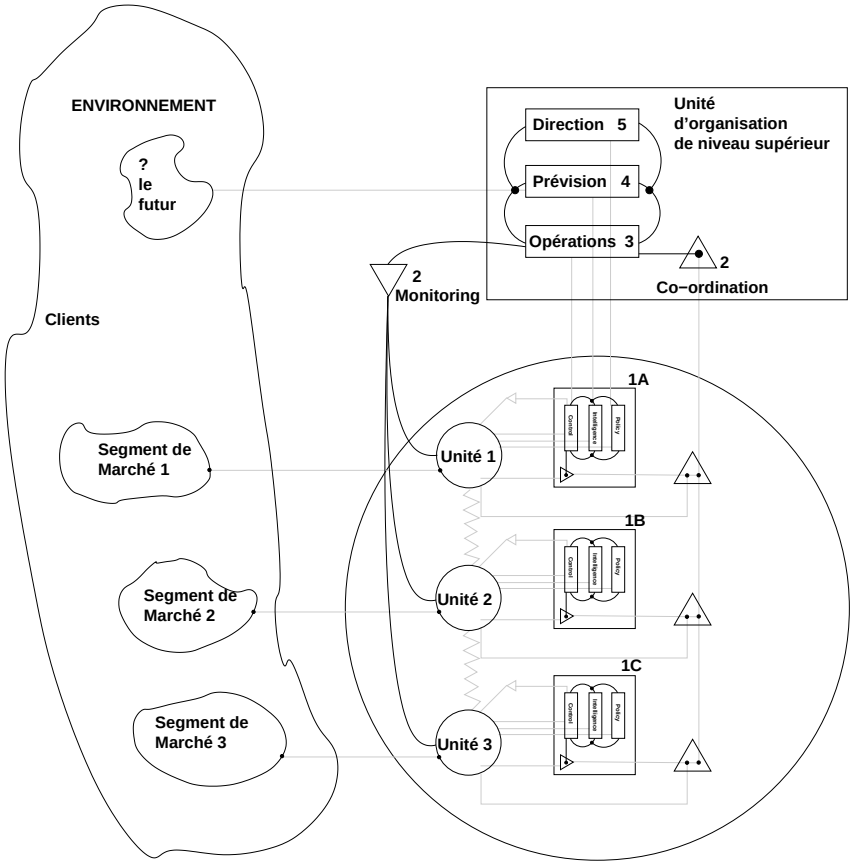

Fig. 2.2 — Diagramme simplifié du *Viable System Model*

On peut distinguer dans cette figure les composants suivants :

5 Niveau direction. La direction règle la conduite globale de l'organisation. Elle fixe les objectifs de haut niveau et contrôle en ce sens le fonctionnement du niveau prévisionnel, mettant en œuvre le patron de conception **Contrôle séparé**.

4 Niveau prévisionnel. Le niveau prévisionnel cherche à définir des lignes de conduite en prenant en compte la direction globale fournie par le niveau direction, ainsi que l'anticipation des changements induits par des perturbations extérieures. Ce niveau constitue donc la partie contrôle du patron de conception **Contrôle séparé**, en fixant les objectifs du niveau opérationnel.

3 Niveau opérationnel. Le niveau opérationnel contrôle directement les composants productifs de l'entreprise. Il est constitué par exemple par les chefs d'équipe. On trouve ici encore un exemple de **Contrôle séparé**.

2 Régulation. C'est le mécanisme de planning de fabrication (pour la coordination), d'audit et d'inspection (pour le *monitoring*), destiné à échanger les informations entre la partie contrôle et la partie processus. Un de ses objectifs est entre autres de prévenir les oscillations.

$\mathbf{1}^{A-C}$ Contrôleurs des niveau hiérarchiques suivants. La même structure est réutilisée aux différents niveaux, le modèle s'appliquant de manière récursive.

Unit 1, Unit2, Unit 3 Les unités de production.

Une des premières constatations est que le contrôle est dissocié de l'objet du contrôle (une usine dans notre cas). On voit apparaître une boucle de rétroaction entre ces deux composants, ainsi qu'entre les éléments constituant ces composants. Le contrôleur 3 (opérationnel) est un contrôleur comme on en trouve dans la théorie des asservissements, qui contrôle directement les actions. Le contrôleur 4 (prévision) est appelé *contrôleur adaptatif*. Il sert à régler le fonctionnement du contrôleur 3. Le contrôleur 5 (direction) est le *contrôleur de supervision*, et contrôle l'interaction entre les contrôleurs 3 et 4.

Herring [Her01, HK00] propose d'appliquer ce modèle à l'architecture de systèmes logiciels, les considérant comme des organisations complexes, afin d'obtenir dans ces systèmes les propriétés équivalentes à celles qu'on peut retrouver dans les organisations, à savoir la *viabilité*. Il prend notamment en compte le fait que les systèmes logiciels sont intégrés et fonctionnent dans un environnement, comprenant entre autres les personnes qui produisent le logiciel ainsi que celles qui l'utilisent selon l'échelle à laquelle on considère le système.

Herring a traduit le modèle de Beer en un langage de patrons de conception. Nous proposons ci-dessous une traduction de son résumé, qui met en relation les différents termes du langage de patrons de conception proposé. On retrouve les mêmes termes dans le tableau 2.1.

> La viabilité est la qualité que présente un système capable de maintenir une existence autonome – i.e. de survivre – à l'intérieur de son environnement. Pour atteindre cette qualité, les systèmes évoluent afin d'obtenir un **Contrôle Séparé** de l'objet du contrôle. L'objet du contrôle est appelé le *processus*. La première responsabilité d'un **Contrôle Séparé** est de s'occuper du **Contrôle Opérationnel** du processus. Cette fonction contrôle

directement le processus en envoyant des commandes, en accordant des ressources pour effectuer les commandes et en recevant périodiquement des informations en retour.

Cependant, le contrôle peut présenter des problèmes : il peut ne pas atteindre l'équilibre du système et se sur-corriger en permanence. Le remède à ce problème est d'introduire un **Centre Régulateur** pour organiser les activités du processus et le coordonner avec d'autres systèmes. Un **Audit Sporadique** est également nécessaire pour fournir des informations nécessaires que le retour standard périodique ne couvre pas.

La deuxième responsabilité du contrôle est d'anticiper sur les futures conditions et de planifier les actions en conséquences. Cette fonction incombe au **Contrôle Adaptatif** du processus. L'interaction de ces deux fonctions, contrôle opérationnel et contrôle adaptatif, donne au processus la capacité d'évoluer pour s'adapter à son environnement.

La dernière exigence du Contrôle Séparé est de rendre le système autonome, en introduisant un **Contrôle de Supervision** qui adopte la politique qui va guider et contraindre les actions du Contrôle Opérationnel et du Contrôle Adaptatif. Des **Alertes** permettent de court-circuiter les couches de contrôle pour informer directement le Contrôle de Supervision de l'occurence d'événements particuliers. Le patron de conception des systèmes viables est alors complet. Il constitue toutefois une base permettant de spécifier comment les systèmes viables sont inclus dans et incluent d'autres systèmes viables dans des chaînes de **Compositions Récursives** similaires. Enfin, chacune des relations entre les systèmes viables et les éléments des systèmes viables est une **Boucle Homéostatique**.

On voit ici que prise sous un angle technique, la définition de la viabilité devient alors :

> «La viabilité logicielle est la qualité d'un système logiciel qui exprime le fait que son architecture peut évoluer d'une adaptation par des opérateurs humains vers une adaptation automatique (sous supervision).»[2] [Her01]

Il est précisé que cette définition inclut les personnes qui produisent le système, étant donné qu'ils sont pour l'instant le seul agent capable d'effectuer une adaptation lors de la conception. Cette définition suppose également que l'évolution des systèmes logiciels a une direction, qui est la création de systèmes intelligents autonomes. Une conséquence de ce postulat est que les opérateurs humains (développeurs, administrateurs, utilisateurs) doivent remplacer ou compenser les fonctionnalités d'autoadaptation incomplètes ou manquantes dans le système, jusqu'à ce que ce dernier les acquière.

[2] « Software Viability is the quality of a software system such that its architecture can be adapted over time by humans (adaptable at design-time) toward becoming an intelligent (supervisory-adaptive- control) system (adaptive at run-time). »

Nous présentons aux paragraphe 2.6.5 la notion de variabilité, dont les auteurs constatent également une loi générale d'évolution des systèmes informatiques qui va dans le même sens que celle énoncée ici mais avec des termes, et donc un point de vue, différents.

Dans son étude des algorithmes génétiques, Holland [Hol92] formule un certain nombre de questions fondamentales qu'il faut se poser dès lors que l'on veut créer un système adaptatif :

- À quelles parties de l'environnement le système s'adapte-t-il ?
- Quelles forces sont exercées par l'environnement sur le système ?
- Quelles sont les structures soumises au processus d'adaptation ?
- Quels sont les mécanismes de l'adaptation ?
- Quelle part des informations sur ses interactions passées avec l'environnement le système conserve-t-il ?
- Quelles sont les limites du processus d'adaptation ?
- Comment peut-on comparer les différents processus d'adaptation ?

Holland identifie de plus trois composants majeurs intervenant dans le processus d'adaptation : E, l'environnement du système sujet à l'adaptation ; τ, le plan d'adaptation décrivant la manière dont l'adaptation est effectuée ; et μ, l'indice de performance, i.e. une mesure de la qualité de l'adaptation du système à son environnement.

On peut faire ici le lien avec le Modèle des Systèmes Viables : on voit apparaître la notion d'environnement, le plan d'adaptation est du ressort du niveau de direction (5), et l'indice de performance est élaboré à partir des informations fournies par le système de régulation (2).

2.2.2 Déclinaisons de la viabilité

Si l'on reprend la définition de viabilité donnée plus haut, l'objectif principal de l'adaptation est d'assurer la survie du logiciel, ce qui consiste en fait en la continuation de son exécution et de son utilisation. Cette viabilité peut dépendre d'un certain nombre de propriétés qu'un système informatique doit conserver.

Assurer le fonctionnement correct du système

Le fonctionnement correct du système, pris dans le sens de sa conformité avec la spécification, doit être assuré en permanence. La complexité du système augmentant avec la prise en compte de multiples alternatives, la possibilité de dysfonctionnement augmente également, à tous les niveaux d'observation considérés.

Ainsi, Netscape, lorsqu'il a changé la licence de son logiciel de navigation du WWW pour produire le logiciel *Open Source Mozilla* [vGBS01], a dû remplacer plusieurs composants propriétaires implémentant des fonctionnalités telles que le déverminage avancé de l'application en cas d'erreur ou la gestion de l'affichage des

fontes. La modularité de l'application est dans ce cas cruciale pour l'adapter aux besoins ainsi induits.

Améliorer la robustesse du système

La robustesse d'un système peut être caractérisée par sa capacité à gérer des situations hors norme, ou encore par son degré élevé de tolérance aux pannes. Le projet Chameleon [BWKI98] vise spécifiquement à fournir une plate-forme distribuée adaptative offrant des services tolérants aux fautes. Pour ce faire, elle offre des services de détection et notification d'erreurs (matérielles, système, applicatives, etc.). Les notifications remontent jusqu'à un composant central appelé FTM (*Fault Tolerance Manager*), qui par le biais de stratégies de tolérance aux fautes spécifiées par l'utilisateur peut modifier le fonctionnement du système afin de s'adapter à la nouvelle situation.

Dans un environnement mobile, le besoin en robustesse découle de la grande variabilité des ressources utilisées [SA00]. Les applications doivent donc être capables de s'accommoder de grandes variations de bande passante, voire de déconnexions complètes, ainsi que d'une autonomie énergétique réduite.

La capacité de redimensionnement du système (*scalability*) peut également être améliorée par l'utilisation de mécanismes autoadaptatifs [MW00], permettant d'utiliser dynamiquement des ressources supplémentaires pour faire face à un accroissement des besoins.

Améliorer les performances

Laddaga [Lad99] indique que « *les logiciels autoadaptatifs fourniront de meilleurs temps de réponse et amélioreront les performances* », grâce au caractère sinon quasi-optimal, du moins plus informé, des choix de conception ou de mise en œuvre.

Par exemple, le programme FFTW [FJ98] (*The Fastest Fourier Transform in the West*) utilise plusieurs techniques d'adaptation afin d'améliorer ses performances. D'une part, il combine de manière dynamique plusieurs fragments de code, appelés *codelets*, en fonction des données à transformer ainsi que des capacités du système. D'autre part, il essaye d'exploiter au mieux la structure mémoire hiérarchique des ordinateurs actuels, en tirant parti des caches processeurs, ou encore du parallélisme offert par des systèmes distribués.

Il offre de plus la possibilité de sauvegarder des informations sur la configuration utilisée, afin de les réutiliser lors des invocations suivantes, afin d'éviter la phase de construction du plan d'exécution si les conditions n'ont pas changé, ou bien de servir de base à la construction d'un nouveau plan d'exécution.

Dans un contexte multimédia, l'adaptation du comportement des applications à la bande passante disponible est un sujet de nombreuses études, que l'on retrouve souvent associées au concept de QoS (*Quality of Service*). Odyssey [NSN+97] par

exemple est une plate-forme permettant de concevoir des applications offrant un affichage des données multimédia adapté aux conditions de fonctionnement. Pour un flux vidéo, il s'agit alors en termes de performances d'assurer un affichage régulier et synchronisé avec le son, au prix de dégradations de la qualité de l'image.

Dans un contexte distribué, l'augmentation des performances du système peut se faire par la migration de certains modules afin d'effectuer un équilibrage de la charge globale [Har98, RSZ87].

2.3 Propriétés de l'adaptation

L'adaptation présente un certain nombre de propriétés essentielles, qu'il faut toujours garder à l'esprit lorsque l'on travaille dans le domaine. La plupart sont des évidences, mais le fait de les expliciter incite à s'y intéresser parituclièrement et à s'assurer qu'elles ont bien été couvertes.

2.3.1 Échelle

L'adaptation peut être considérée à différentes échelles. Comme le modèle de Beer le souligne, les systèmes sont par nature récursifs, et peuvent être observés à différents niveaux. À chaque niveau, on peut percevoir différemment l'échelle de temps, ainsi que les structures et données manipulées, ce que Van Gurp [vGBS01] exprime en parlant de représentations spécifiques à chaque sous-système.

2.3.2 Temporalité

L'adaptation est un processus temporel. L'analyse de systèmes adaptatifs doit prendre en compte cet aspect dynamique. On ne doit pas se contenter d'une vision statique, à quelque niveau que ce soit.

Dynamicité

L'adaptation induit une dynamicité, plus ou moins perceptible suivant l'échelle où l'on observe le système. Holland [Hol92] précise : « Dans tous les cas, les échelles de temps peuvent être librement réinterprétés dans différentes applications – ce peut être des nanosecondes dans une application, des siècles dans une autre. »[3]

Par exemple, lors de l'adaptation d'un système dans sa modélisation (que l'on qualifie également d'évolution), l'échelle de temps se mesure en mois, voire en années.

Dans le cas d'une adaptation lors de l'exécution, le principe est de coller aux besoins de l'application, ce qui peut se traduire par des temps de réaction de l'ordre

[3] « In any case, the instants of time can be freely reinterpreted in different applications – they may be nanoseconds in one application, centuries in another. »

de la seconde.

Bien entendu, les contraintes imposées par ces vitesses de réaction hétérogènes sont diverses, et reflètent les besoins en outils et techniques propres à chacun des niveaux : sur une échelle de temps longue, comme lors de la modélisation, ce sont plutôt des besoins d'organisation et de conservation des informations historiques qui sont nécessaires. Dans les échelles de temps plus courtes, telles les contraintes du temps réel, les besoins techniques sont plus importants.

Durées d'adaptation

On peut distinguer plusieurs données concernant la vitesse d'adaptation, représentées dans la figure 2.3. Les deux principales sont la durée de la phase d'adaptation, ainsi que la durée de la phase stable.

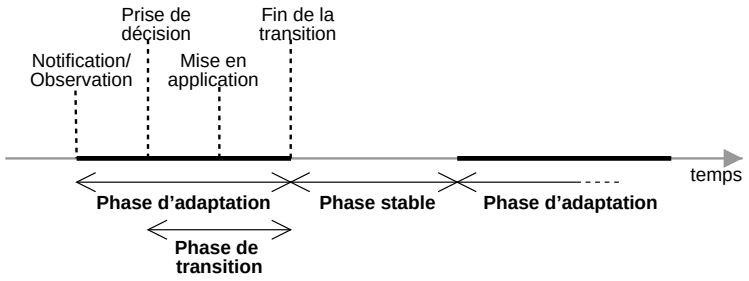

Fig. 2.3 — Durées d'adaptation

On considère la *phase d'adaptation* comme l'intervalle de temps entre l'observation ou la notification d'une observation entraînant un besoin de changement, et la fin du processus d'adaptation. Cette phase comprend une *phase de transition*, qui est la période durant laquelle l'ancienne configuration n'est pas encore invalidée, et la nouvelle configuration non encore autonome. À la fin de la période de transition, le système est dans un état cohérent.

La *phase stable* est la période de fonctionnement standard du logiciel, durant laquelle aucune adaptation n'est nécessaire.

La durée de la phase stable ne doit pas être nulle. Il convient donc de s'assurer que durant la phase d'adaptation, la notification éventuelle de nouveaux changements est prise en compte d'une manière ou d'une autre. Cette prise en compte peut être tout simplement d'ignorer les nouveaux changements, au prix d'une possible incohérence.

Noble *et al.* [NSN+97] définissent la notion d'agilité, représentant la vitesse et la pertinence des détections et des réponses aux changements dans la disponibilité des ressources.

2.3.3 Stabilité

Cette propriété pourrait tout aussi bien être décrite par son contraire – l'instabilité. Dès lors que l'on introduit des mécanismes dynamiques, il est important d'assurer la stabilité du système, notamment par l'établissement de diverses protections assurant que les conditions limites de fonctionnement sont toujours atteintes.

Divers moyens sont mis en œuvre pour assurer cette stabilité : hystérésis [HMS99], fonctionnement sous-optimal, politiques génériques, etc.

Une inspiration pourrait venir du domaine de recherche des systèmes autostabilisants (*self-stabilizing systems*), dont l'idée a été proposée par Dijkstra [Dij74] : « Un système est autostabilisant quand indépendamment de son état initial, le système arrivera inévitablement à un état légitime en un nombre fini d'étapes. » Ce domaine de recherche concerne les système distribués. Les algorithmes de routage utilisés sur IP comme OSPF ou RIP constituent par exemple des systèmes autostabilisants.

2.3.4 Multiplicité

Afin de pouvoir assurer une adaptation du fonctionnement d'un système, il faut disposer de multiples alternatives. Dans le cas d'une adaptation de paramètre, cette multiplicité est naturellement atteinte. Dans le cas d'une adaptation de comportement ou d'architecture, les choix sont plus restreints et doivent être identifiés. La première étape consiste à fixer l'échelle à laquelle on désire travailler. Ensuite, posant une des questions fondamentales de Holland présentées au paragraphe 2.2.1, il faut identifier les structures soumises au processus d'adaptation. Enfin, l'énumération des alternatives possibles peut être effectuée. Tekirnedogan [AT99, Tek96] propose par exemple une algèbre permettant d'énumérer l'ensemble des alternatives, afin ensuite de manipuler divers scénarios.

On peut enfin imaginer des manières de créer cette multiplicité. Les travaux autour de la notion de spécialisation et d'évaluation partielle [JGS93] permettent d'obtenir diverses stratégies adaptées à des conditions particulières d'utilisation.

2.3.5 Observation - Introspection

Le processus d'adaptation s'effectue en regard d'un environnement (d'exécution, de modélisation, etc.). Il est donc nécessaire, afin de juger de la pertinence de l'adaptation et également d'en contrôler les mécanismes, d'acquérir des informations sur ledit environnement.

Ces informations viennent de l'observation de l'environnement extérieur au système considéré, mais également de capacités d'introspection intégrées au système lui-même. Un système adaptatif est en effet partie intégrante de son environnement, et à ce titre participe aux changements qui y surviennent.

Ces deux pendants de l'observation se retrouvent dans le modèle défini dans le

paragraphe 2.1 et représenté dans la figure 2.1 : l'observation doit être effectuée sur la partie pertinente de l'environnement, ainsi que sur les données importantes du système lui-même, ce qui donne le *contexte* du système.

Il importe également, comme le souligne entre autres Shaw [Sha95], de bien distinguer l'information relative à l'environnement, utilisée par le mécanisme d'autoadaptation, des informations utilisées pour la fonction elle-même. Dans certains cas, elles peuvent être amenées à se recouper, comme par exemple dans le cas des algorithmes de tri autoadaptatifs qui prennent en compte la nature des données à trier.

2.4 Types d'adaptation

À quelles parties du système peut-on appliquer l'adaptation ? On rejoint ici une des questions fondamentales de Holland [Hol92] rappelée dans le paragraphe 2.2.1. Dans le cadre de cette étude, on distingue principalement trois types d'adaptation : l'adaptation de paramètres, l'adaptation de comportements et l'adaptation d'architectures.

2.4.1 Adaptation de paramètres

L'adaptation de paramètres est celle la plus couramment rencontrée. La théorie des asservissements utilise en effet majoritairement cette notion, afin de maintenir certains paramètres constants.

L'utilisation du paradigme de contrôle a été utilisé notamment dans les domaines de traitement de signal, pour concevoir par exemple des systèmes d'annulation d'écho ou d'égalisation adaptatifs [Jr95]. Ces systèmes utilisent une estimée de l'erreur en sortie du système en tant que paramètre de l'algorithme. La plupart de ces algorithmes sont caractérisés par un besoin d'excitation constante, afin d'éviter une dérive de l'estimation ou l'apparition de pics dans le signal.

Shaw [Sha95] considère ce type d'adaptation lorsqu'elle utilise le modèle de contrôle des processus pour l'appliquer à des problèmes informatiques.

2.4.2 Adaptation de comportements

Au delà de la simple modification de paramètres numériques du système, on peut envisager un changement complet de stratégie. Les stratégies deviennent donc, typiquement par le biais d'une réification, des paramètres du système.

Ainsi, la plate-forme de création de serveurs web adaptatifs Jaws [HS99, Hu98] offre deux types d'adaptation de comportements. L'adaptation statique permet de sélectionner l'algorithme le plus adapté aux exigences statiques du système, comme par exemple une gestion de la concurrence implémentée différemment selon que le

système est mono- ou multi-processeur. L'adaptation dynamique offre par ailleurs la possibilité d'adapter dynamiquement un algorithme durant l'exécution du serveur.

À un autre niveau, on peut trouver une adaptation de comportements lors de la compilation d'applications permettant de sélectionner lors de la configuration un comportement dans un ensemble. Par exemple, l'éditeur `emacs` [Sta94] offre lors de sa compilation la possibilité de choisir entre deux interfaces graphiques différentes (Motif ou Athena Widgets).

De multiples techniques peuvent être utilisées pour accomplir ce type d'adaptation. Nous en présentons quelques unes au chapitre 4. Cependant, quelle que soit la technique utilisée, on peut dégager une structure ou un mécanisme commun. Le patron de conception *Stratégie Autoadaptative*, présenté dans le chapitre 3, fournit un cadre reprenant les différents éléments des systèmes à adaptation de comportements afin de permettre d'analyser et de mettre en œuvre de tels systèmes.

2.4.3 Adaptation d'architecture

Le principe du changement de comportement repose sur la modification d'un composant, mais au sein d'une architecture fixe. Il est également possible d'envisager un changement d'architecture afin d'adapter plus largement la structure du système.

On trouve naturellement cette approche dans la phase d'analyse et de modélisation d'un système informatique. Par contre, les travaux autour de l'adaptation dynamique d'architecture sont plus novateurs et moins répandus [MSL00, FPC00, OGT+99, Tay96].

Oreizy *et al.* [OGT+99] isolent particulièrement deux modèles de description d'architectures dynamiques : C2 [Tay96] et Weaves [GR91]. Ce sont deux modèles d'architecture prenant en compte un aspect dynamique. Ils partagent plusieurs points communs :

- ils sont tous les deux fondés sur la notion de composants et de connecteurs ;
- ils ne placent aucune restriction sur la granularité des composants ou leur langage d'implémentation ;
- ils requièrent tous deux une communication asynchrone entre les composants ;
- les composants peuvent encapsuler des fonctionnalités d'une complexité arbitraire.

Cependant, les deux projets diffèrent dans leur approche de la composition de systèmes. C2 construit les systèmes selon une organisation hiérarchique, où un composant n'est conscient que des couches supérieures et ne possède aucune connaissance des composants de même niveau ou des couches inférieures, communicant avec ces derniers par le biais de connecteurs. Weaves adopte de son côté une approche tournée vers le flux des données et du séquencement du système. Ainsi, les composants ne font qu'accepter des données en entrée et produire des données en sortie, sans savoir d'où elles viennent ou ce qu'il en advient. C'est le rôle des connecteurs que d'ajouter une sémantique par la liaison des différents composants.

Ce type d'adaptation concerne également la mobilité. En effet, dans le cadre des réseaux mobiles, il peut être nécessaire de migrer une fonctionnalité d'un site vers un autre, ce qui revient à adapter l'architecture du système distribué à la configuration courante.

2.5 Les moments d'adaptation

Boinot [BMN+00] propose une classification des systèmes adaptables au sens large selon deux axes : un axe représentant le moment d'injection du comportement, et un autre représentant le moment de liaison. Nous avons représenté ces deux axes sur la figure 2.4, et y avons placé les exemples que nous utilisons par la suite pour illustrer les différents types d'adaptation.

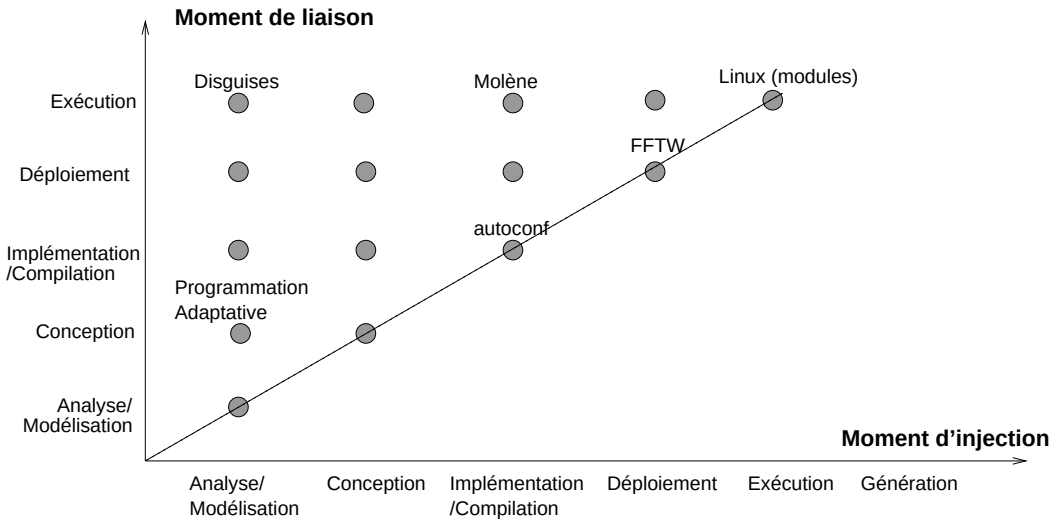

FIG. 2.4 — Classification

Le *moment d'injection* du comportement est l'instant auquel un comportement est incorporé au système. On parcourt donc une échelle allant de la conception à l'exécution. On peut notamment distinguer l'injection lors de la conception/compilation, l'injection dynamique de comportements existants lors de l'exécution, et en dernier lieu la génération automatique de comportements par le système.

Le *moment de liaison* est l'instant où l'on pratique l'adaptation, où l'on choisit le comportement adéquat. De manière similaire au moment d'injection, on retrouve une échelle allant du plus statique au plus dynamique : liaison lors de la conception (compilation en dur de comportements), liaison lors du début de l'exécution (configuration du système ou liaison dynamique), liaison au cours de l'exécution (adaptation dynamique).

La combinaison de ces deux axes permet d'aider à préciser l'échelle à laquelle on considère l'adaptation.

Van Gurp [vGBS01] propose une notion similaire, qu'il qualifie de *points de variation*. Dans une optique de développement logiciel, il identifie les points où l'on est amené à prendre des décisions de conception se rapportant aux fonctionnalités désirées ou à des exigences de qualité.

Une des thèses développées est la suivante : la tendance actuelle dans les développements de systèmes informatiques est de repousser au maximum le moment de cette prise de décision. Ainsi, un choix d'implémentation que l'on aurait pu faire au moment de la conception peut être repoussé au moment du chargement de l'application. On peut assimiler cette idée à la loi d'évolution du logiciel présentée dans le paragraphe 2.2.1. Cependant, cette échelle tend à linéariser les deux dimensions présentes dans la classification selon les moments d'injection et de liaison.

2.5.1 Analyse / Modélisation

L'adaptabilité logicielle représente la faculté d'évolution du code d'un système informatique. C'est un paradigme de programmation destiné à prendre en compte le fait que le code ainsi que les structures de données utilisés dans un programme sont amenés à évoluer au cours du temps.

Il est donc nécessaire de prendre en compte dès le départ cette possible évolution, et de faire en sorte que les changements soient les plus simples possibles. Dans cette optique, on va chercher à minimiser les dépendances entre classes, ou entre algorithmes et structures de données.

Le principe d'adaptation est donc appliqué ici à la conception de l'application plutôt qu'à son comportement. Nous aborderons dans le paragraphe 2.6.2 les travaux de Tekirnedogan [AT99, AT98, Tek96] autour de l'adaptabilité au , ainsi que ceux de van Gurp [vGBS01] sur la variabilité au paragraphe 2.6.5. Ces travaux proposent une méthodologie et des outils permettant d'obtenir une adaptation dès la modélisation du système, même s'ils peuvent être utilisés pour couvrir une étendue plus large des moments d'adaptation. La programmation adaptative [GH91], utilisation spécialisée de la programmation par aspects [KLM+97a], permet également d'introduire des concepts d'adaptation lors de la phase de modélisation.

La programmation par aspects

La programmation par aspects [KLM+97a] est une nouvelle méthodologie de programmation qui vise à obtenir une séparation maximale des différents aspects d'un programme (*separation of concerns*).

Par aspect, on entend une vue particulière sur le système que l'on veut concevoir. Des aspects classiques peuvent être des politiques de synchronisation, des méthodes de distribution, des stratégies de gestion d'erreurs, etc.

La caractéristique d'un aspect est qu'il n'est pas possible de l'exprimer par des unités de structure classiques comme des modules. En effet, il étend sa fonctionnalité

sur plusieurs modules de manière transversale à la structure du système.

La programmation par aspects fournit donc une nouvelle unité de modularité qui permet d'encapsuler des concepts transversaux aux unités classiques de modularité tels que les objets ou les composants. La programmation par aspects introduit des mécanismes explicites permettant d'exprimer ces modularité transversales. Ainsi, leur structure apparaît plus clairement, et il est plus simple de les manipuler. Ce mode d'expression exprime également plus clairement les interactions entre le système initial et les aspects qu'on y applique.

La programmation adaptative

Un cas particulier de la programmation par aspects est la *programmation adaptative* [Lie00], dans laquelle un des aspects est exprimé sous la forme d'un graphe de blocs de construction, et les autres aspects ou composants utilisent des *stratégies* pour contrôler le parcours et interpréter les références à ces graphes. Un principe essentiel est la séparation la plus forte possible entre le code et les données (*structure-shy code*).

La programmation adaptative (ou *Adaptive Programming (AP)* est un cas particulier de programmation par aspect (*Aspect-Oriented Programming (AOP)*) dans lequel les blocs de construction peuvent être exprimés sous la forme de graphes, et où les autres blocs de construction utilisent ces graphes grâce à des stratégies de parcours (*traversal strategies*) [LPS97].

Une stratégie de parcours peut être considérée comme une spécification partielle d'un graphe définissant quelques nœuds et arêtes de base. Une stratégie de parcours découpe les graphes sur lesquels elle est appliquée, en ne mentionnant qu'un nombre limité de nœuds et d'arêtes. Les stratégies de parcours peuvent être considérées comme des expressions régulières spécifiant un parcours à travers un graphe. (Formellement, une stratégie de parcours pour un graphe G est un sous-graphe de la fermeture transitive de G auquel on a ajouté de l'information).

En résumé, AP (Programmation Adaptative) = AOP (Programmation par Aspects) avec des graphes et des stratégies de parcours. En application de cette distinction, on peut dire que la programmation adaptative est un cas particulier de programmation par aspects, où une partie du découpage de l'application est exprimée de manière variable en utilisant une stratégie qui isole des graphes plus petits à l'intérieur de grands graphes. Une des fonctionnalités essentielles de cette isolation est qu'elle est spécifiée en cachant les détails des grands graphes. Les isolations sont ainsi *adaptatives* en ce sens qu'elles fonctionnent pour une grande variété de grands graphes.

Plus concrètement, la programmation par aspects incite à diviser un projet en différents blocs (nœuds de graphe) couvrant divers aspects d'un système. Il peut arriver que pour répondre à un besoin d'un aspect on dispose de plusieurs solutions différentes, donc de plusieurs nœuds de graphes équivalents mais distincts. La

programmation adaptative va définir à l'intérieur du graphe d'interconnexion des différents nœuds plusieurs sous-graphes correspondant chacun à une alternative de combinaison des nœuds.

Le modèle *Disguises*

Les exemples cités précédemment (programmation par aspect et programmation adaptative) relèvent du niveau modélisation à la fois pour le moment d'injection et pour le moment de liaison. Le modèle *Disguises* [SHMP98, SJJ+00] tente d'adopter une démarche permettant d'effectuer lors de la modélisation une séparation de différents aspects du système, ainsi qu'une adaptation dynamique de ces aspects lors de l'exécution du système.

Les différents aspects isolés, appelés *disguises*, relèvent de la synchronisation, de la concurrence ou de la distribution, et sont spécifiés dans un langage dédié. Un langage de composition permet ensuite de modifier dynamiquement les différentes politiques lors de l'exécution. Ce modèle se place donc dans notre classification dans un moment d'injection lors de la modélisation et un moment de liaison lors de l'exécution.

2.5.2 Compilation

Un des exemples canoniques d'adaptation lors de la compilation est illustré par l'utilisation du préprocesseur dans le langage C. Les directives présentes permettent de spécifier l'inclusion ou la non-inclusion de certaines parties du programme, ainsi que des comportements différents suivant la plate-forme utilisée ou les fonctionnalités présentes dans le système.

On peut constater même à ce niveau une progression vers une autonomie plus grande du système, avec l'introduction par le projet GNU d'un outil appelé *autoconf* [MMF94], qui permet d'effectuer une détection automatique des fonctionnalités présentes et de la plate-forme. Ainsi, l'adaptation lors de la configuration peut s'effectuer, sur ces critères tout du moins, sans requérir d'assistance humaine.

2.5.3 Configuration / Déploiement

Une adaptation que l'on qualifie parfois de statique peut avoir lieu lors du déploiement du système, dans une phase de configuration. Dans ce mode, diverses alternatives sont choisies en fonction des conditions d'exécution mesurées ou constatées au démarrage. Cette approche offre un compromis souvent intéressant entre un besoin de s'adapter à des conditions de fonctionnement et la complexité de mise en œuvre d'une adaptation dynamique lors de l'exécution.

L'application de calcul de transformées de Fourier FFTW [FJ98], déjà mentionnée dans le paragraphe 2.2.2, offre un exemple de ce type d'adaptation. Lors de l'initialisation de l'application, un plan d'exécution est établi en fonction des carac-

téristiques du système et de la taille de la transformée discrète. Le plan ainsi calculé définit la combinaison de *codelets* (petits morceaux de codes dédiés) la plus efficace au niveau temps de calcul. Les auteurs font d'ailleurs remarquer que le temps de calcul et la complexité algorithmique sont peu corrélés, notamment à cause de l'impact des caches processeur.

2.5.4 Exécution

On peut distinguer plus précisément les deux axes de classification (injection et liaison) pour le moment d'exécution.

Moment d'injection

Le système d'exploitation *Linux* fournit un système de modules qu'il est possible d'insérer dynamiquement au sein du noyau. Les pilotes de périphériques sont un des domaines privilégiés de l'utilisation de ce système. Ce mécanisme a été étendu pour permettre de modifier dynamiquement l'ordonnanceur de processus d'un système actif [Rhi00].

Barnes et Pandey [BP99] proposent avec leur langage spécialisé *CacheL* de permettre à des serveurs web de spécifier des politiques spécifiques de remplacement, cohérence, etc., aux caches web qui pourraient être amenés à conserver une copie de leurs documents. Des comportements nouveaux sont donc injectés dans des caches web au cours de leur exécution. Dans le même ordre d'idées, Pierre *et al.* [PKvST00] adoptent le point de vue qu'une politique générique ne peut être efficace au sein d'un cache web, et qu'il est donc important de permettre l'application de politiques spécifiques à chaque document.

Moment de liaison

L'application d'audioconférence *freephone* [BFPVG97] est un exemple simple de ce cas de figure. Plusieurs algorithmes de compression des données audio sont présents dans l'application, et chacun est adapté à un débit particulier. En fonction de l'observation de la bande passante disponible, l'application choisit d'utiliser un des algorithmes en particulier.

Dans le domaine des réseaux sans-fils, l'application de prescription médicale en milieu hospitalier Charcot, développée dans le cadre du projet MolèNE [SA00], est capable de modifier son mode d'accès à une base de données en fonction de la qualité de la connexion au réseau sans-fil.

Enfin, la plate-forme Jaws [HS99, Hu98] de conception de serveurs web permet d'adapter diverses politiques aux conditions d'exécution. Par exemple, la politique de gestion de la concurrence, qui détermine comment le serveur gère les différents fils de

contrôle, est modifiable dynamiquement en cours d'exécution, suivant l'observation des ressources disponibles.

Les coûts

Il existe une grande variété de techniques permettant de mettre en œuvre une adaptation dynamique des systèmes logiciels. Cependant, les concepteurs d'applications peuvent chercher à éviter d'avoir à les utiliser, pour plusieurs raisons :

- Elles ne sont souvent pas nécessaires. L'adaptation au moment de l'exécution n'est pas considérée comme un aspect critique des systèmes informatiques, et on peut trouver des techniques alternatives, telles que des périodes d'arrêt régulières, de la redondance ou du fonctionnement en grappes, ou encore de la maintenance manuelle.
- Les risques sont augmentés. Il est plus difficile d'assurer l'intégrité, la fiabilité et la robustesse du système dans le cadre de changements en cours d'exécution.
- Les coûts sont augmentés. Le surcoût en termes de performances est généralement sensible lors de l'utilisation de mécanismes d'adaptation en cours d'exécution [DSC+99]. De plus, le manque de techniques adéquates pour modéliser et concevoir des systèmes dynamiquement adaptables augmente les coûts de conception.

2.6 Déclinaisons de l'adaptation

Comme présenté dans le paragraphe 2.2, l'adaptation d'un système est une des conditions à sa viabilité. Diverses approches ont été développées pour traiter de cette adaptation, introduisant des termes spécialisés. Le choix de ce vocabulaire n'est pas innocent et peut amener à des points de vue différents sur les manières de considérer l'adaptation. Les différents critères que nous avons énoncés dans les paragraphes précédents vont nous permettre de préciser un certain nombre des propriétés de chaque type d'adaptation, afin en particulier de s'assurer de l'adéquation du vocabulaire utilisé par rapport aux intentions.

Dans un premier temps, nous allons donc indiquer la manière dont on peut exprimer les différents critères présentés précédemment afin de spécifier un mode d'adaptation particulier. Nous verrons ensuite comment on peut par ce biais qualifier différents termes traitant tous d'adaptation, mais de manières différentes. Ainsi, l'**adaptabilité** (voir paragraphe 2.6.2), qui exprime une capacité d'adaptation, est rendue tangible par la conception de systèmes **adaptatifs** (voir paragraphe 2.6.4). L'adaptabilité y est traduite via divers moyens relevant de la **variabilité** (voir paragraphe 2.6.5) ou de la **configurabilité** (voir paragraphe 2.6.6) du système. Nous allons préciser dans la suite de ce paragraphe les significations que nous accordons à ces différents termes, en les reliant aux travaux qui les ont introduits ou formalisés.

2.6.1 Qualification de l'adaptation

Afin de lever un certain nombre d'ambiguïtés, il est important lorsque l'on traite de l'adaptation de préciser systématiquement un certain nombre de critères. Sur la base des propriétés présentées dans la première partie de ce chapitre, nous proposons une qualification qui réponde systématiquement aux trois questions essentielles :

- Quelle est la structure soumise à l'adaptation ?
- À quel moment de la vie du système a lieu l'adaptation ?
- Qui effectue l'adaptation ?

Dans une version synthétisée, on retrouvera une désignation contenant les réponses à ces trois questions. Ainsi, une des acceptions du terme *configuration* peut être précisée par la désignation suivante :

> Configuration = adaptation par l'*utilisateur* d'un *comportement* au moment du *déploiement* du système.

Dans le cadre d'un échange avec plusieurs interlocuteurs, cette spécification permet de s'affranchir des incompréhensions provenant d'interprétations différentes du même terme. Nous allons voir dans la suite de ce chapitre quelques exemples de termes relatifs à l'adaptation, que nous préciserons par notre dénomination.

2.6.2 Adaptabilité

L'adaptabilité représente la capacité d'adaptation que présente un système, c'est-à-dire sa faculté de modifier son fonctionnement en fonction de ses conditions d'exécution. Cette capacité est introduite dans un système par les concepteurs du système. Comme le soulignent Aksit et Tekinerdogan [AT99], fournir une adaptabilité maximale peut créer un surcoût lors de l'exécution. Le travail des concepteurs de systèmes est donc de comparer, d'évaluer et de décider de l'architecture à utiliser parmi les différentes alternatives, à la lumière des exigences de qualité.

Tekinerdogan [Tek96, AT99] propose un formalisme, appelé algèbre de conception (*design algebra*), qui permet de décrire un ensemble d'alternatives d'implantation possibles pour une architecture, ainsi qu'un outil de génération et de comparaison des modèles possibles. Cette algèbre est basée sur l'identification des concepts fondamentaux d'un domaine. Une fois identifiés, il est possible de les représenter dans une notation ensembliste et d'automatiser des manipulations permettant d'énumérer des configurations possibles suivant les contraintes que l'on fixe.

Ainsi, l'analyse du domaine du tri d'ensembles [AT98] fait apparaître les concepts suivants : l'algorithme de tri proprement dit Alg, la dimension de l'espace à trier D, les opérations de lecture/écriture RW et le critère de tri CR. L'ensemble $S = (Alg, D, RW, CR)$ définit un algorithme. L'adaptabilité est définie comme l'ensemble $A = (FX, AD)$ où FX représente les concepts fixés du système, et AD les concepts adaptables. Le tri adaptatif AS est alors défini comme le produit cartésien

des deux ensembles $AS = SxA = (Alg, D, RW, CR)x(FX, AD)$. Par des manipulations algébriques, il est alors possible d'identifier des configurations plus ou moins adaptables.

Afin de concorder avec les changements du contexte, il est nécessaire d'adapter le système. Puisque le système et l'environnement important constituent tous deux le contexte, un changement de contexte signifie un changement dans l'un des ou dans les deux éléments.

On considère que le contexte inclut les *éléments variables* du système et de l'environnement, c'est-à-dire les éléments dont on peut attendre un changement. Ce changement peut également prendre la forme d'un remplacement ou d'une suppression. Les éléments variables sont donc compris dans le contexte. La notion symétrique est celle d'invariant.

Si l'on postule qu'un domaine est composé d'un ensemble d'éléments variables et d'invariants, nous pouvons décrire le système par les formules suivantes, illustrées par la figure 2.5 :

$$\text{domaine} \subset \text{éléments variables} \cup \text{invariants}$$

$$\text{éléments variables} = \text{éléments variables du système} \cup \text{éléments variables de l'environnement}$$

$$\text{invariants} = \text{invariants du système} \cup \text{invariants de l'environnement}$$

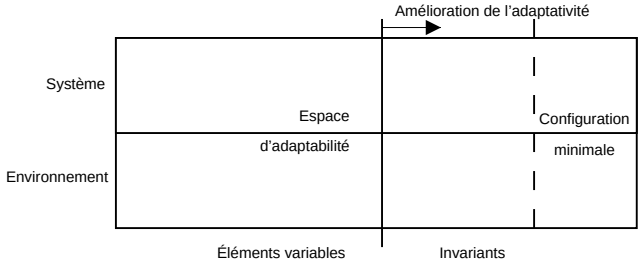

FIG. 2.5 — L'espace d'adaptabilité

Tekinerdogan introduit alors la notion d'espace d'adaptabilité, qui est constitué par les éléments variables qui peuvent être modifiés. L'adaptabilité est principalement réduite à cause du fait qu'un certain nombre des éléments variables possibles sont considérés comme des invariants du système. Il est donc possible d'améliorer l'espace d'adaptabilité en identifiant les éléments possiblement variables dans l'ensemble des invariants. Néanmoins, cet espace d'adaptabilité ne peut pas être étendu à l'infini. Le système doit en général disposer d'une *configuration minimale* sur laquelle il va se baser. Le configuration minimale du système est l'ensemble des invariants nécessaires au fonctionnement du système. Si la configuration minimale du système est atteinte, on peut dire que le système est *optimalement adaptable*. Il est alors possible de l'adapter en fixant les valeurs des éléments variables en fonction des contraintes.

Pris dans ce contexte, l'adaptation est appliquée aux comportements possibles que l'on peut trouver dans le système. Le concepteur du système introduit dans le système les mécanismes permettant d'adapter les comportements dans un instant se situant entre la conception du système et son exécution, sous la supervision d'une entité non précisée. On peut donc définir l'adaptatibilité comme une potentialité d'adaptation de *comportements*.

2.6.3 Flexibilité

Un autre terme, synonyme d'adaptabilité, est celui de flexibilité [Cah96, BMN+00]. On le trouve souvent utilisé lorsque le terme d'adaptabilité est réduit à l'adaptabilité dynamique (en cours d'exécution), et que l'on veut pouvoir désigner l'ensemble des autres domaines de l'adaptabilité, notamment au niveau de la conception d'un système. On peut donc préciser ici que la flexibilité est la potentialité d'adaptation de *comportements* lors de *l'exécution*.

2.6.4 Adaptativité

L'adaptativité traduit le fait que l'adaptabilité d'un système a été exprimée. Ainsi, comme le souligne Stefani [Mul97] :

> «
> – un système adaptable offre la possibilité de se configurer en fonction des besoins d'une application ;
> – un système adaptatif possède la capacité algorithmique de se configurer en fonction d'un environnement qui change.
> Un système adaptable permet d'écrire plus facilement des applications adaptatives. »

Dans ce contextel la structure soumise à l'adaptation n'est pas précisée. Cependant, on peut identifier les deux autres critères de notre dénomination : l'adaptation a lieu lors de l'exécution, et le système l'effectue de manière autonome. On peut donc en tirer que l'adaptativité, dans ce cas, exprime une adaptation d'une structure par le *système* lors de son *exécution*.

2.6.5 Variabilité

La variabilité est un concept synonyme de l'adaptabilité, développé par Van Gurp et Svahnberg [vGBS01]. Les auteurs postulent également l'existence d'une loi générale gouvernant l'évolution des systèmes logiciels : les décisions de conception concernant les fonctionnalités ou les exigences de qualité sont prises de plus en plus tard dans le cycle de vie du logiciel. Ils constatent de plus qu'au cours du développement, le nombre de systèmes potentiels diminue jusqu'à ce que l'on arrive à un système unique lors de l'exécution.

La variabilité représente donc la capacité du système à être implémenté de diverses manières. On peut identifier dans un système des *points de variation*, qui sont les emplacements où l'on doit prendre, à un moment ou à un autre, une décision de conception.

Les auteurs de [vGBS01] proposent ainsi une terminologie adaptée à la description de cette variabilité, ainsi qu'une classification des points de variation et une notation graphique permettant de les représenter sous la forme d'un graphe de fonctionnalités étendu. Un exemple en est donné dans la figure 2.6.

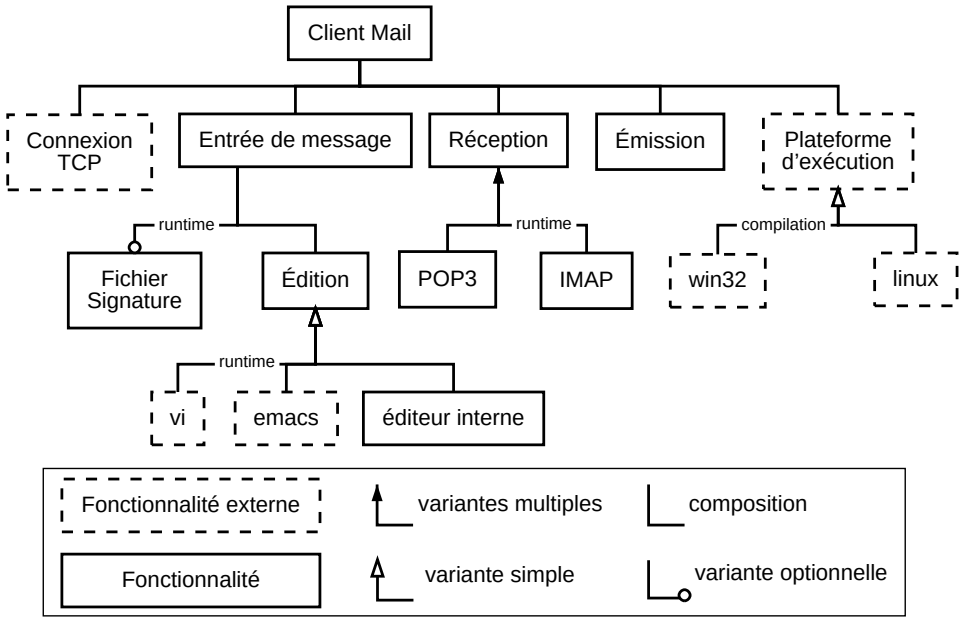

FIG. 2.6 — Graphe de fonctionnalités pour un client de courrier électronique

Les points de variation sont identifiés en particulier par le niveau auquel ils se présentent : conception architecturale, conception détaillée, code source, code compilé, code lié et système en cours d'exécution.

Chaque phase de développement possède sa propre représentation, que l'on peut considérer comme un niveau d'abstraction. Le développement d'un système est donc assimilable à une suite de transformations de ces représentations. Lors de chaque transformation, des décisions de conception sont prises, et d'autres sont laissées ouvertes pour être résolues plus tard.

Les auteurs proposent de plus trois patrons généraux permettant de préciser un type de variation :

- variante simple. Dans ce cas, il existe un ensemble de variantes. Au moment du choix, une seule des variantes est choisie.
- variante optionnelle. C'est une spécialisation de la variante simple, où l'ensemble des variantes contient un seul élément, dont l'utilisation est optionnelle.

- variantes parallèles multiples. Lorsque des variantes parallèles sont utilisées, le point de variation n'est pas lié à une variante de manière permanente, mais le processus de choix est effectué à chaque fois que le point de variation est utilisé.

Enfin est présentée une méthodologie pouvant servir de guide lors de l'élaboration de systèmes variables :

- identification des endroits où la variabilité peut être nécessaire ou utile ;
- applications de contraintes : choix d'un moment de résolution, choix du moment et de la manière d'ajouter des variantes, choix d'un patron général de variabilité, choix d'une représentation ;
- implémentation de la variabilité, en utilisant des techniques appropriées ;
- gestion des variantes.

Le concept de variabilité présenté ici couvre donc un spectre assez large de moments d'adaptation, allant de la conception du système jusqu'à son exécution. L'entité effectuant l'adaptation varie également, en fonction du moment d'adaptation considéré : l'adaptation lors de la conception est effectuée par le concepteur du système, tandis que l'adaptation lors de l'exécution peut être pratiquée par le système lui-même. Le concept non-variant de cette adaptation est ici la structure soumise à l'adaptation : les comportements. On peut donc préciser que la variabilité traite de la possibilité d'adaptation de *comportements* à différents momens de la vie du système.

2.6.6 Configurabilité

La configurabilité relève d'un moment particulier dans le cycle de vie d'un système informatique, celui de son chargement. Cependant, de nombreuses personnes, dont Oreizy *et al.* [OGT+99], utilisent ce terme pour qualifier un mode particulier d'adaptation s'effectuant au niveau de l'architecture des applications, comme nous l'avons vu dans le paragraphe 2.4. Dans cette dernière acception, la configurabilité du système représente donc la capacité d'adaptation par le *système* de l'*architecture* du système lors de son *exécution*.

2.7 Conclusion

Nous avons ici voulu identifier l'objectif principal de l'adaptation, qui est la *viabilité* du système informatique concerné. Cette propriété générale, issue d'un modèle du domaine de la cybernétique qu'il est possible de transposer dans le domaine informatique, peut s'exprimer plus concrètement sous diverses formes : fonctionnement correct du système, amélioration des performances, augmentation de la robustesse.

Nous avons ensuite identifié une partie des propriétés fondamentales communes à tout niveau d'adaptabilité : l'importance de l'échelle d'analyse ; l'aspect temporel et dynamique de l'adaptation ; les problèmes de stabilités induits par la dynamicité

introduite ; le besoin de la présence d'alternatives ; et enfin l'importance de l'introspection. Puis nous avons précisé les trois grands types d'adaptation que sont l'adaptation de paramètres, l'adaptation de comportements et l'adaptation d'architecture. Enfin, nous avons présenté une classification temporelle de l'adaptation selon deux axes, qui distinguent le moment d'injection de comportements du moment de liaison.

Ces différentes propriétés nous permettent de proposer une dénomination plus précise des différents modes d'adaptation. Pour illustrer cette dénomination, nous l'avons appliquée à différents termes tous relatifs à l'adaptation, et vu comment ils s'articulaient entre eux : l'adaptabilité, comme la flexibilité, représente la capacité d'adaptation d'un système, l'adaptativité d'une application est la traduction de cette adaptabilité. La variabilité est un concept similaire à celui d'adaptabilité, mais construit autour d'un formalisme particulier, les points de variation. Enfin, la configurabilité peut, selon le contexte, désigner une adaptation ayant lieu à un instant particulier de la vie de l'application, ou bien une adaptation s'appliquant au niveau architectural. L'ensemble de ces approches a pour propriété commune d'augmenter le degré d'autonomie des systèmes, allant ainsi dans le sens d'une loi globale d'évolution des systèmes informatiques.

Cette classification nous permet également de préciser le cadre de développement du patron de conception que nous proposons dans le chapitre suivant, et de limiter sa portée à l'étude de l'adaptation dynamique de comportements déjà existants (moment d'injection avant l'exécution, moment de liaison à l'exécution), comme représenté par le rectangle grisé sur la figure 2.7.

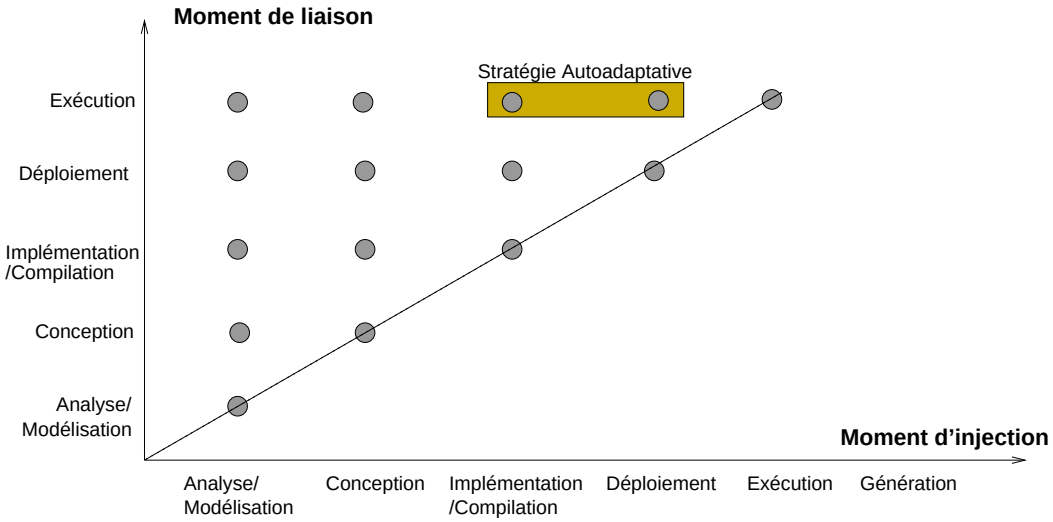

Fig. 2.7 — Placement du patron de conception *Stratégie Autoadaptative*

Nous avons donc vu qu'il existe divers types d'adaptation, que l'on distingue notamment suivant la structure du système soumise à l'adaptation, reprenant ainsi une des questions fondamentales posées par Holland et mentionnée au paragraphe 2.2.1. Dans la perspective de mettre en œuvre l'adaptation dans des caches web, nous avons

identifié dans le chapitre 1 plusieurs fonctionnalités qui offrent une grande variété algorithmique. Nous nous sommes donc orientés plus précisément vers l'étude de l'adaptation de comportements. Dans ce cadre, nous avons cherché à identifier les mécanismes les plus importants à mettre en place. Il s'est avéré que de nombreux systèmes adaptatifs existent déjà, et que l'on peut en dégager une structure commune. C'est pourquoi nous avons orienté notre travail vers la définition d'un patron de conception.

CHAPITRE 3

Le patron de conception *Stratégie Autoadaptative*

Le concept d'adaptation peut se décliner de différentes manières, notamment en fonction des structures du système soumises au processus d'adaptation. L'adaptation des comportements – ou stratégies – d'un système peut facilement être effectuée lors d'une phase de compilation ou de modélisation, où l'on choisira une stratégie plutôt qu'une autre. Repousser ce choix jusqu'au moment de l'exécution du système requiert la mise en œuvre de mécanismes spécifiques. Des solutions existent, déclinées sous différentes formes : support du langage, plate-forme de conception, mécanismes offerts par le système d'exploitation, etc. On peut toutefois isoler un certain nombre d'éléments communs parmi ces solutions. L'expression de ces éléments communs permet non seulement de comprendre plus facilement le fonctionnement des systèmes adaptatifs existants, mais également de fournir des guides lors de la conception de nouveaux systèmes adaptatifs. Nous proposons ici la définition du patron de conception *Stratégie Autoadaptative* afin d'organiser la réflexion sur les systèmes adaptatifs. Une version condensée du patron de conception est présentée dans l'annexe A.

Dans un premier temps, nous présentons la notion de patron de conception en l'illustrant par quelques exemples. Sur la base d'un exemple d'application adaptative, nous identifions dans le paragraphe 3.3 les différentes étapes du processus d'adaptation et décrivons la structure du patron de conception que nous proposons, qui permet de mettre en œuvre deux types d'adaptation au sein d'un système. Le paragraphe 3.4 étudie les conséquences qui découlent de l'utilisation de ce patron de manière générale, tandis que le paragraphe 3.5 se penche sur les enjeux de mise en œuvre que l'on peut trouver dans chacun des composants proposés. Nous décrivons enfin (paragraphe 3.6) la manière dont on peut envisager de combiner les deux types d'adaptation au sein d'un même système, ainsi que la possibilité d'application récursive du patron à ses propres composants (paragraphe 3.7).

3.1 Les patrons de conception

Les patrons de conception indiquent, pour un contexte donné, une solution éprouvée déjà rencontrée dans d'autres systèmes similaires. Cette solution ne peut se restreindre à un simple algorithme, cette approche étant trop rigide pour se généraliser facilement. On utilise donc un formalisme qui permet de spécifier le contexte du problème et les enjeux présents. Ces éléments étant précisés, la solution proposée peut être décrite et les conséquences, tant positives que négatives, de sa mise en œuvre peuvent être indiquées. On permet ainsi le partage d'une expertise dans un domaine particulier.

D'autre part, outre cet aspect de transmission de savoir, les patrons de conception permettent également de créer un vocabulaire spécifique à un domaine. À cet effet, chaque patron de conception a un nom, de préférence évocateur. Ainsi, les experts du domaine disposent de notions plus abstraites dédiées à leur domaine, ce qui facilite les discussions et la description de solutions. L'approche des patrons de conception est donc un effort communautaire pour capitaliser l'expérience acquise dans un domaine et faciliter sa propagation.

Nous allons brosser un rapide historique du concept de patron, puis préciser la manière dont il a été transposé dans le domaine informatique. Nous traiterons ensuite d'un des enjeux relatif à la représentation des patrons et à leur traitement automatisé par divers outils. Enfin, à titre d'illustration, nous présenterons quelques patrons issus de [GHJV95], en les utilisant pour analyser deux exemples d'applications : un noyau de système d'exploitation et une application Web d'interrogation de base de données.

3.1.1 Historique

À la fin des années 80, Ward Cunningham et Kent Beck ont voulu trouver un moyen de guider des programmeurs novices dans le domaine de la conception d'interfaces utilisateur en Smalltalk [BC87]. Ils ont pour ce faire utilisé la notion de patron de conception, concept introduit plusieurs dizaines d'années auparavant par l'architecte Christopher Alexander. Ce dernier cherchait à documenter les archétypes de conception qu'il avait pu identifier dans son domaine de travail. Pour ce faire, il a proposé une manière de présenter ces informations afin d'en faire ressortir les éléments les plus importants, et de les rendre plus facilement transmissibles.

De nombreux informaticiens, principalement issus du domaine de la programmation orientée objet, ont cherché à transposer ces idées dans le domaine de la conception de systèmes informatiques, afin de documenter à leur tour les pratiques existantes d'un domaine qui pouvait être considéré comme naissant [App97].

Nous allons revenir succinctement sur l'origine (non informatique) de la notion de patron et son développement consécutif dans le domaine informatique. Nous préciserons ensuite les caractéristiques des patrons informatiques, relatives notamment à leur caractère récurrent, et verrons quels sont les éléments qui les constituent. Enfin,

nous nous intéresserons aux modes de représentation des patrons, ainsi qu'aux outils de conception ou d'analyse qui les exploitent.

L'origine des patrons de conception

La réflexion de Christopher Alexander [Ale79, Ale77] a pour origine un besoin de concevoir des lieux d'habitation et de travail plus adaptés aux besoins de leurs utilisateurs. Il a introduit la notion de patron de conception comme un cadre permettant d'exprimer les conséquences qu'il a tiré de ses expériences. Dans [Ale77], il décrit des idées architecturales *atemporelles* permettant d'atteindre ces objectifs. Il raffine ensuite ce cadre [Ale79] en proposant un paradigme architectural orienté autour de trois concepts : la *Qualité*, le *Moyen* et la *Manière*.

La *Qualité* représente l'objectif final à atteindre lors de la construction d'une ville ou d'un immeuble. Elle englobe différents paramètres tels que le confort, l'habitabilité, la résistance, la capacité d'adaptation, etc.

Le *Moyen* est le mécanisme permettant d'atteindre cette *Qualité*. Il peut être exprimé sous la forme d'un langage de patrons, dont l'utilisation peut conduire à la construction d'architectures répondant à la *Qualité* désirée. Il représente l'ensemble des savoirs que l'on peut trouver dans le domaine de l'architecture.

Les différents patrons présents dans le *Moyen* sont combinés selon la *Manière* – qualifiée également de *manière atemporelle* (*Timeless Way*), ce qui nous ramène au titre de son ouvrage [Ale79] – afin de concevoir par raffinements successifs une construction répondant à la *Qualité* désirée.

Ainsi, en combinant les patrons du *Moyen* suivant une certaine *Manière*, il est possible d'atteindre la *Qualité* désirée.

De plus, un patron de conception, dans le sens où l'entend Alexander, n'est pas une simple représentation abstraite d'une connaissance acquise dans un domaine. Cette représentation, par son explicitation, déclenche un mécanisme génératif permettant de créer à nouveau des entités répondant aux même canons. Ainsi, chaque patron constitue une règle décrivant ce qu'il faut faire pour générer l'entité qu'il définit. Alexander [Ale79] indique qu'*un patron est à la fois une entité, que l'on peut observer, et la règle qui nous indique comment créer cette entité, et à quel moment on doit l'utiliser. C'est donc à la fois un processus et une entité ; à la fois la description d'une entité dynamique, et la description du processus qui permet de générer cette entité.*

La transposition dans le domaine informatique

Les analogies possibles entre le domaine de l'architecture de bâtiments et le domaine de l'architecture de logiciels ont amené quelques informaticiens à se pencher sur le cadre de réflexion proposé par Alexander. Comme il arrive souvent, la même idée était développée indépendamment par plusieurs personnes. Ainsi, Erich

Gamma [Gam91] réfléchissait au même moment, à la fin des années 1980, sur la manière de représenter les structures de conception récurrentes dans la conception de systèmes informatiques. La rencontre de ces différentes personnes, à travers les conférences TOOLS (*Technology of Object-Oriented Languages and Systems*) et OOPSLA (*Object-Oriented Programming, Systems, Languages, and Applications*), leur a permis de confronter leurs idées et de constater la convergence de leurs points de vue.

La suite logique de ce constat était de favoriser la diffusion des idées ainsi développées. C'est ainsi que quatre personnes – Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides – ont réuni dans un ouvrage [GHJV95] une première liste documentée des patrons de conception qu'ils avaient pu isoler. Cet ouvrage constitue une des références de la communauté des patrons de conception. De nombreux choix peuvent en être discutés, mais un des objectifs des patrons de conception étant de créer un vocabulaire commun permettant d'échanger des idées à propos de la conception de système informatiques, il est important de reconnaître la base que constitue ce livre. Il a introduit des concepts qui sont maintenant largement utilisés, tels que celui de *Strategy* (Stratégie)[1]. ou *Observer* (Observateur). Cet ouvrage est souvent identifié par le nom de *GoF*, abbréviation de *Gang of Four*, en référence à ses quatres auteurs.

3.1.2 Clarification de la notion de patron

Que peut-on trouver dans un patron de conception utilisé dans un contexte informatique ?

Tout d'abord, un patron représente une idée *utile, utilisable et utilisée*. Le premier terme manifeste l'idée qu'un patron de conception doit effectivement résoudre un problème donné dans un contexte donné. Le second terme traduit le caractère pratique d'un patron de conception : il indiquent la méthode de résolution d'un problème, mais donne également la manière de construire cette méthode. Enfin, le dernier terme, *utilisée*, souligne le fait qu'un patron de conception documente une solution déjà éprouvée à plusieurs reprises. Il n'est pas une création originale, mais l'explicitation d'une solution déjà observée. Ce dernier point ne diminue cependant pas l'aspect créatif des patrons de conception : trouver le bon niveau d'abstraction pour exprimer un mécanisme existant peut requérir autant, voire plus, d'efforts que concevoir le mécanisme lui-même.

Dans sa structure, un patron de conception peut être considéré comme une règle

[1]Nous utilisons sciemment dans ce document les dénominations anglo-saxonnes des patrons de conception, pour la raison suivante : les patrons de conception sont censés, entre autres, fournir un vocabulaire commun à un domaine particulier. Il est important que ce vocabulaire soit largement partagé. Une traduction des noms de patrons de conception courants, tels que ceux définis dans [GHJV95], peut introduire une confusion, surtout en cas de traductions différentes d'un même nom, et restreindre l'intérêt de l'utilisation des patrons. De plus, la concision que permet la langue anglaise dans la qualification des patrons n'est pas forcément toujours possible en français.

en trois parties, qui exprime une relation entre un contexte, un ensemble de forces (ou contraintes) que l'on retrouve régulièrement dans ce contexte, et une certaine architecture logicielle qui permet de résoudre les contraintes reconnues.

Types de patrons

On peut distinguer plusieurs niveaux d'application pour les patrons utilisés dans le domaine informatique, ce qui conduit à identifier divers types de patrons [BMRaMS96].

Patrons architecturaux Un patron architectural présente un schéma ou une structure fondamentale pour un système informatique. Il propose un ensemble de sous-systèmes, précise leurs fonctionnalités et définit la manière dont sont organisées leurs inter-relations.

Patron de conception Un patron de conception fournit un schéma permettant de préciser le fonctionnement ou les relations de sous-systèmes ou de composants d'un système infromatique. Il décrit les structures récurrentes permettant de résoudre un problème de conception dans un contexte donné.

Idiome Un idiome – que l'on peut également qualifier de *patron de programmation* – est un patron de plus bas niveau, spécifique à un langage de programmation. Un idiome indique la manière de mettre en œuvre un aspect particulier d'un composant en utilisant des fonctionnalités spécifiques au langage.

Comme on peut le constater, ces différents types de patrons ne sont pas indépendants les uns des autres : par exemple, un idiome est l'expression d'un patron de conception pour un langage particulier, dans un contexte plus précis. Ils partagent tous les éléments essentiels des patrons : le contexte, les forces mises en évidence dans le problème et le modèle de solution, plus ou moins précis suivant le niveau considéré.

Dans une autre dimension d'analyse, les auteurs de [GHJV95] distinguent trois types de patrons, selon leur caractère plus ou moins dynamique. Ainsi, les patrons peuvent décrire plus particulièrement la structure d'organisation de différents composants, le caractère générateur d'un système ou la manière dont il se comporte. Pour chacun des patrons décrits dans l'ouvrage est donc précisé la catégorie à laquelle il se rapporte : *structural*, *creational* ou *behavioral*.

Expression d'un patron

On peut utiliser différents formalismes pour exprimer un patron de conception. Les auteurs précisent parfois explicitement le formalisme qu'ils utilisent, sous le nom de *forme Alexandrienne* ou *forme GoF*. La première se rapporte aux ouvrages de Alexander [Ale79, Ale77, Ale64] et à sa manière d'exprimer les patrons de conception dans le domaine de l'architecture, qui incite notamment à utiliser des illustrations

graphiques d'utilisations effectives du patron. Le format *GoF* est celui utilisé dans l'ouvrage [GHJV95] (voir section 3.1.1), qui consacre une partie bien identifiée par un label à chacun des éléments le composant.

Malgré cette diversité dans la mise en forme, on retrouve dans la quasi-totalité des patrons les mêmes éléments essentiels que nous allons maintenant décrire. Ces éléments sont soit précisés explicitement dans un patron de conception, soit immédiatement identifiables.

Nom Un patron est identifié par un nom évocateur. Ce nom permet de faire référence de manière rapide et efficace au patron, et donc aux informations qui y sont rattachées. On constitue ainsi un vocabulaire d'abstractions facilitant les échanges entre les acteurs du domaine considéré. On peut parfois trouver plusieurs synonymes pour le même patron, qui sont alors référencés comme tels.

Problème Cet élément exprime de manière succinte les objectifs que le patron de conception cherche à atteindre, à l'intérieur du contexte et du système de forces et de contraintes précisés.

Contexte Le contexte décrit le domaine d'application du patron. Il précise les conditions requises pour l'application du patron, identifiées par l'analyse des instances déjà existantes du patron.

Forces/Contraintes Cet élément décrit les différentes forces et contraintes interagissant au sein du contexte précisé. Elles précisent les différents enjeux du problème posé, et permettent de déterminer les compromis qui doivent être envisagés.

Solution On décrit dans cet élément la manière de mettre en œuvre la solution proposée, tant d'un point de vue statique (structure) que dynamique (interactions). Cette description peut être sous différentes formes (texte, schéma, etc), et présente notamment les différentes entités qui sont présentes dans la solution, la manière dont elles sont agencées ainsi que la manière dont elles interagissent. La description du patron peut donner des indications à garder en mémoire lors de l'application du patron, que ce soit des guides de mise en œuvre ou des mises en gardes sur certains points. On peut également trouver une description de variantes de la solution.

Exemples Un ou plusieurs exemples d'application du patron, qui permettent d'illustrer le contexte d'application, la manière de lui appliquer le patron et la façon dont cette application transforme le contexte. Les exemples sont destinés à clarifier les conditions d'application et la manière d'utiliser le patron. Ils peuvent être accompagnés d'un exemple simple de mise en œuvre, pour montrer une des façons de réaliser le patron. Un patron étant l'expression d'une solution déjà mise en œuvre dans plusieurs systèmes, on utilise souvent des exemples tirés d'applications existantes.

Contexte résultant On décrit ici le contexte résultant de l'application du patron au contexte initial. Cette description expose notamment les conséquences – positives et négatives – de l'utilisation du patron, ainsi que les nouveaux enjeux

qui se posent. Il est ainsi possible d'utiliser un nouveau patron dont le contexte initial correspondra au contexte résultant du patron de conception courant.

Justification La justification du patron de conception indique les raisons fondamentales qui conduisent à l'utilisation de la solution proposée, et en quoi cette solution est pertinente. Elle constitue une réflexion sur les qualités du patron.

Patrons associés On indique par ce biais les patrons de conception qui peuvent être associés au patron courant. Les patrons associés partagent souvent certaines forces ou contraintes. De plus, leur contexte initial ou résultant peut correspondre au contexte résultant ou initial d'un autre patron. Enfin, ils peuvent constituer une alternative au patron considéré, prenant en compte d'autres aspects du problème ou effectuant des compromis différents.

Utilisations connues Une des conditions nécessaires (mais non suffisante) de la validité d'un patron est son utilisation dans plusieurs systèmes existants. Un patron doit en effet être une solution utilisée pour résoudre un problème récurrent. Les utilisations connues du patron sont souvent utilisées pour fournir des *exemples* d'application.

Langages de patrons

Il est possible de combiner un certain nombre de patrons en un ensemble cohérent, que l'on qualifie alors de *langage de patrons*. La conception d'un système dans un domaine particulier met en effet en jeu de nombreux patrons. Chacun de ces patrons traite d'un problème particulier relatif à la conception du système envisagé. Un langage de patrons fournit alors un guide architectural, indiquant notamment quels sont les patrons les plus intéressants à utiliser, et comment ils s'articulent entre eux. Cette notion de langage de patrons est également issue des travaux de l'architecte Alexander [Ale77].

Un parallèle peut être effectué avec la notion de cadre de conception (*framework*) que l'on connait déjà dans le domaine informatique. Un patron de conception propose un certain nombre de briques logicielles, que l'on complète afin d'obtenir un système fonctionnel. Un langage de patrons offre de la même manière un certain nombre de briques (sous la forme de patrons), articulées de manière cohérente. L'utilisateur de ce langage dispose alors d'informations le guidant dans la conception du système envisagé.

3.1.3 Outils de manipulation des patrons

Les patrons de conception fournissent un formalisme pour exprimer des idées. Leur popularité croissante dans le monde informatique pose des enjeux relatifs à leur mode de diffusion et d'utilisation. En effet, les patrons étant destinés à constituer un moyen de capitaliser et transmettre une expérience, il importe de s'assurer de leur accessibilité. Cet enjeu pose de plus la question de la représentation des patrons dans

l'objectif de leur transmission. Les points essentiels présentés au paragraphe 3.1.2 précisent les éléments constitutifs d'un patron, mais non leur représentation.

De plus, une fois que l'on dispose d'un patron dans une représentation précise, des outils permettant leur manipulation, en particulier leur mise en œuvre, deviennent intéressants. Inversement, on peut vouloir extraire de manière plus ou moins automatique des patrons de conceptions depuis un système existant.

Représentation des patrons

La base cruciale de ces enjeux repose sur la manière de représenter les patrons de conception. On dispose d'ores et déjà de représentations textuelles telles que la *forme Alexandrienne* ou la *forme GoF* décrites dans le paragraphe 3.1.2. Cependant, ces représentations ne sont pas conçues dans un objectif de manipulation automatique. Il est donc difficile de concevoir des outils d'extraction permettant d'assister un concepteur dans sa recherche de patrons pertinents relatifs au problème qu'il traite.

Des efforts de recherche sont actuellement portés sur la représentation graphique de la structure des patrons de conception, accompagnée éventuellement d'informations complémentaires.

Une extension du méta-modèle UML [GSJ00] a ainsi été proposée, qui étend le concept de collaboration ainsi que le langage de spécification de contraintes OCL afin de mieux pouvoir représenter des patrons de conception dans un schéma UML. L'objectif principal de ce travail est d'intégrer ensuite cette notation dans un outil de modélisation UML, afin de pouvoir disposer de la notion de patron au niveau de la modélisation et de pouvoir profiter des fonctionnalités de génération automatique de code.

Le projet LePUS (*Language for Pattern Uniform Specification*) [Ede02] propose une notation graphique permettant de représenter la structure d'un patron ainsi que certaines des contraintes qui y sont associées. L'intégration de cette notation dans le logiciel PatternWizard permet la génération automatique de code correspondant à la mise en œuvre de patrons.

Les tentatives de formalisation actuelles s'orientent donc plus particulièrement vers l'expression de la solution proposée par le patron. Les autres éléments du patron (décrits dans la section 3.1.2), dont le contexte et l'énumération des forces en jeu, ne sont généralement pas traités. Cette orientation se justifie si l'on considère que l'enjeu principal considéré actuellement est l'utilisation des solutions proposées par les patrons à l'intérieur d'outils de modélisation et de génération de code. Cependant, les outils d'assistance à la recherche de patrons adéquats pour un contexte donné ne sont pas encore réellement envisagés.

Diffusion des patrons

Un des objectifs des patrons est la constitution d'un vocabulaire d'abstraction, rendant plus aisée et plus concise la communication d'idées entre les personnes impliquées dans un domaine particulier. Il est nécessaire, à partir du moment où l'on désire faire communiquer des personnes entre elles, de faire en sorte que les mots qu'elles utilisent représentent bien les mêmes idées. On a donc besoin d'une référence, l'équivalent d'un dictionnaire spécialisé du domaine considéré.

Une des références premières pour les patrons du domaine informatique est constituée par l'ouvrage [GHJV95], qui a constitué un vocabulaire de patrons largement reconnu aujourd'hui. Les notions de *Strategy* (Stratégie) ou d'*Observer* (Observateur) (que l'on rappelle brièvement au paragraphe 3.1.4) constituent maintenant une base commune pour de nombreuses personnes.

Une tentative d'inventaire des patrons nouvellement présentés est menée par le groupe Hillside, qui édite les actes (éponymes) des conférences *Pattern Languages of Programming*. Ces ouvrages sont destinés à garder une trace des patrons présentés (et validés) lors de ces conférences, et constituent également une référence souvent utilisée.

Une tentative plus informelle de diffusion des patrons existants a été lancée par Cunningham, à partir du constat que le mode d'écriture des patrons se prêtait bien à une représentation hypertexte. Il a donc créé un site Web nommé le *Portland Pattern Repository* [Cuna], destiné à rassembler les descriptions des patrons les plus reconnus, et à être enrichi par tous ses utilisateurs par le biais d'un système d'édition collaboratif nommé *Wiki* [Cunb].

Ces différentes solutions participent donc à la diffusion du nouveau vocabulaire introduit par les patrons. Cependant, la profusion d'informations rend difficile leur appréhension. Des outils d'aide à la recherche de patrons de conception pourraient donc être développés afin d'assister un utilisateur dans la résolution d'un problème.

Mise en œuvre des patrons

Une fois que l'on a identifié un patron de conception intéressant, il faut ensuite le mettre en œuvre. Cette mise en œuvre peut s'effectuer de manière plus ou moins automatisée, grâce au support d'un certain nombre d'outils [BFVY96].

Le langage de mise en œuvre peut intégrer des éléments permettant de faire référence directement à certains patrons. Ainsi, OpenJava [TC98] étend le langage de programmation Java pour fournir des méthodes directes d'instanciation de patrons. L'intégration à ce niveau des patrons assure leur identification, mais impose l'utilisation d'un langage spécifique. De plus, elle restreint la gamme des patrons utilisables à ceux qui peuvent être assimilés à des idiomes de programmation.

L'introduction de la notion de patron au niveau d'outils de modélisation de systèmes informatiques, en travaillant à un niveau d'abstraction supérieur, permet d'uti-

liser une gamme plus large de patrons de conception, pour peu qu'une représentation adéquate en soit disponible. Par exemple, la plate-forme UMLaut [GSJ00] intègre la notion de patron et permet de manipuler des modèles UML la prenant en compte, générant ainsi le code correspondant au patron désiré.

Détection automatique de patrons

Enfin, les patrons permettent de comprendre la manière dont fonctionne un système informatique. Ils peuvent résumer de manière synthétique les interactions parfois complexes intervenant entre différents composants du système. Cependant, pour pouvoir bénéficier de cette connaissance, il faut que les patrons sont explicités. On peut distinguer deux façons pour ce faire.

La première consiste à documenter le système informatique lors de son développement en précisant dans la documentation elle-même les différents patrons utilisés ainsi que les composants sur lesquels ils s'appliquent. La maintenance future du système peut se trouver facilitée par cette explication de plus haut niveau.

La seconde méthode peut s'appliquer sur des systèmes existants, n'ayant pas fait l'objet d'un tel effort de documentation. Il est alors possible de leur appliquer des outils de détection automatique d'occurence de certains patrons – simples la plupart du temps – qui apporteront une aide précieuse lors de l'analyse du système en vue de sa modification ou de sa rétro-conception. Le système PTIDEJ [GJ01, AACGJ01] permet ainsi d'analyser une application écrite dans le langage Java et d'en dégager les utilisations de patrons. Pour cela, il analyse l'application pour en extraire un modèle enrichi, et applique sur ce dernier un système de contraintes qui cherchera à détecter les contraintes correspondant à un certain nombre de patrons.

3.1.4 Exemples de patrons

Afin d'illustrer l'utilisation des patrons de conception, nous allons en détailler quelques uns, issus de l'ouvrage [GHJV95]. À la suite de leur présentation, nous les mettrons en évidence dans deux exemples de systèmes informatiques présentant des caractéristiques différentes : un système d'exploitation tel que Linux et une application Web d'interrogation de base de données. Nous désirons ainsi montrer le caractère générique de l'approche des patrons de conception.

La présentation des patrons que nous effectuons ici suivra une mise en forme inspirée du format *GoF* (voir le paragraphe 3.1.2). Pour chacun des patrons, nous indiquerons l'objectif qu'il cherche à atteindre, son contexte d'utilisation, sa structure générale (statique et dynamique), ainsi que certaines informations relatives à sa mise en œuvre, dont les conséquences de son utilisation. Différents exemples d'utilisation sont détaillés à la suite de l'exposé des différents patrons, afin de mettre en valeur le caractère assez étendu du domaine d'application.

Le patron de conception *Adapter* (Adaptateur)

Objectif Le patron *Adapter* permet de convertir l'interface d'une classe en une autre convenant mieux aux besoins du client. Elle facilite ainsi la réutilisation de composants existants, en permettant de mettre à jour leur interface.

Contexte Ce patron est souvent utilisé dans le cadre d'une réutilisation d'anciennes classes, dont l'interface ne correspond plus aux besoins des applications plus récentes. Cette réutilisation pourrait se faire par le biais d'une réécriture complète d'une nouvelle classe, mais cette démarche peut augmenter le temps de développement, introduire de nouveaux bogues, etc. L'utilisation d'un adaptateur permet d'effectivement réutiliser des composants existants au comportement éprouvé, en limitant l'effort de développement et les probabilités d'introduire de nouvelles erreurs.

Structure On distingue pour ce patron deux modes de fonctionnement, selon le mécanisme utilisé pour le mettre en œuvre : l'héritage ou la composition.

L'*adaptateur de classe* utilise le mécanisme d'héritage multiple, en héritant à la fois de la classe à adapter et de la classe définissant la nouvelle interface.

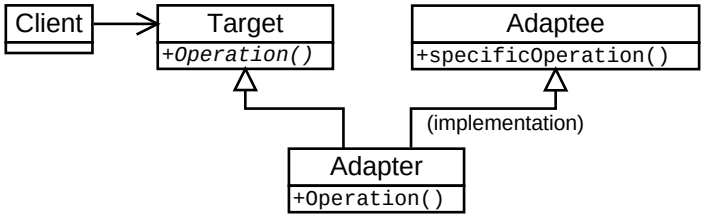

FIG. *3.1* — Patron de conception *Adapter* (héritage)

L'*adaptateur d'objet* utilise le mécanisme de composition, et fonctionne au niveau d'une instance. Cette dernière va maintenir une référence vers une instance de l'objet à adapter, et lui transmettre les requêtes après d'éventuelles transformations.

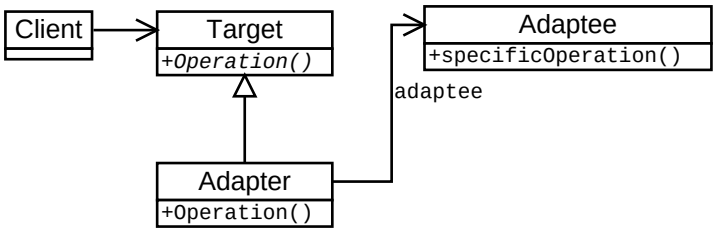

FIG. *3.2* — Patron de conception *Adapter* (composition)

Collaborations Dans les deux cas de mise en œuvre, l'adaptateur (classe Adapter) retransmet les requêtes reçues du Client vers la classe adaptée (Adaptee). La diffé-

rence entre les deux mises en œuvre provient de la manière dont l'adaptateur accède aux méthodes de la classe adaptée : dans le premier cas (adaptateur de classe), le mécanisme d'héritage fournit un accès direct par la classe Adapter aux méthodes adéquates. Dans le second cas (adaptateur d'objet), on utilise le mécanisme de composition : l'instance de la classe Adapter contient une référence sur une instance de la classe adaptée, et transmet explicitement les appels de méthodes.

Mise en œuvre Le choix d'un des deux modes du patron *Adapter* entraîne des compromis différents. En résumé, un adaptateur de classe permet de surcharger des méthodes originales de la classe adaptée, mais ne généralise pas facilement l'adaptation aux sous-classes. L'adaptateur d'objet permet par contre d'adapter l'interface d'une classe et de ses sous-classes.

De plus, suivant l'étendue des différences entre l'interface d'origine et l'interface de destination, la complexité de l'adaptateur peut varier d'une simple conversion de nommages jusqu'à la création de nouvelles opérations non présentes auparavant.

Le patron de conception *Bridge* (Passerelle)

Le patron *Bridge* est similaire dans sa structure au patron *Adapter* présenté plus haut. Tous deux favorisent la flexibilité du code en introduisant un niveau d'indirection supplémentaire. Cependant, l'intention et le contexte d'application diffèrent. En effet, le patron *Adapter* s'applique à un système *après* qu'il ait été conçu, dans l'objectif de le faire fonctionner dans un nouveau contexte non prévu lors de sa conception. Le patron *Bridge* est lui pris en compte *lors* de la conception du système, afin de lui apporter des caractéristiques de modularité et d'extensibilité.

Objectif Le patron *Bridge* promeut le découplage entre une abstraction et sa mise en œuvre, afin que chacune puisse être modifiée indépendamment de l'autre. Alors que le patron *Adapter* est utilisé pour adapter *a posteriori* l'interface de classes existantes, le patron *Bridge* est utilisé dès la conception du système pour introduire un découplage entre interface et implémentation.

Contexte Ce patron est, comme on l'a noté plus haut, utilisé lors de la conception d'un nouveau système afin d'améliorer sa capacité de s'accomoder de changements. Il permet d'éviter une liaison trop forte entre une abstraction et sa mise en œuvre, de manière à pouvoir sélectionner la mise en œuvre adéquate lors de l'exécution du système par exemple. Une même mise en œuvre peut être utilisée par différents objets, sans que ce soit visible du point de vue des clients. De plus, des modifications de mise en œuvre au niveau des objets ne doivent pas entraîner de recompilation des clients, grâce au respect de l'interface.

Structure - Collaborations Généralement, l'interface de la mise en œuvre fournit uniquement des méthodes simples, tandis que l'abstraction définit des opérations de plus haut niveau utilisant ces méthodes simples. Un changement d'implémentation n'aura pas d'impact sur l'interface proposée au client via l'abstraction.

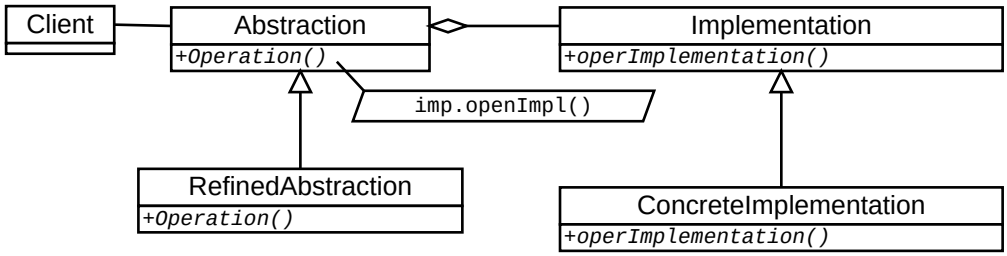

FIG. *3.3* — Patron de conception *Bridge*

Le patron de conception *Facade* (Façade)

Objectif Le patron de conception *Facade* peut être perçu comme l'extension du patron de conception *Bridge* à un système plus complexe : il fournit une interface unifiée à un ensemble d'interfaces disponibles dans un système, évitant ainsi au client d'avoir à connaître le détail des différents sous-systèmes.

Contexte La modularisation des systèmes informatiques apporte de nombreux avantages, mais entraine également des inconvénients tels que l'augmentation de la complexité dans l'utilisation de ces systèmes. Il est donc important de faciliter l'utilisation d'un système complexe, en fournissant des points d'entrée simples et synthétiques pour les utilisations les plus courantes du système. Les utilisateurs du système sont ont alors la possibilité d'accéder au système via l'interface simple, pour les besoins courants, ou bien de s'adresser directement aux éléments individuels pour des besoins plus spécifiques.

Cette manière de procéder apporte de plus, comme dans le cas du patron de conception *Bridge*, une plus grande indépendance du client par rapport au système, ce qui permet de faire évoluer ce dernier sans impliquer de réécriture du client.

Enfin, le patron de conception *Facade* permet, dans une architecture en couches, de définir plus précisément la portée de chacune des couches en lui donnant un point d'accès précis.

Structure - Collaborations Le point d'entrée (*Facade*) sait à quels sous-systèmes sont destinées des requêtes particulières, et peut donc les transmettre au sous-système adéquat. Le client ne s'occupe alors que d'envoyer le message, sans se préoccuper de la destination finale. La *Facade*, comme dans le cas du patron *Bridge*, peut faire suivre

les messages directement, ou bien les transformer de manière à les faire correspondre à ce qu'attendent les sous-systèmes.

Les sous-systèmes n'ont de leur côté pas de connaissance de la *Facade*, afin de permettre leur utilisation indifféremment par le biais de la *Facade* ou bien directement par le client.

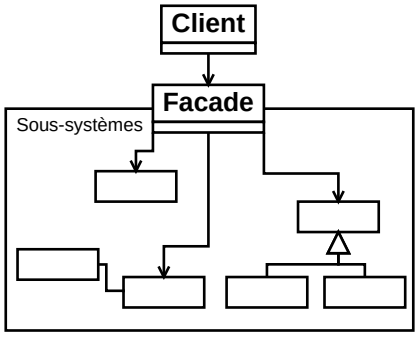

FIG. 3.4 — Patron de conception *Facade*

Le patron de conception *Singleton* (Singleton)

Objectif Le patron de conception *Singleton* assure l'unicité de l'instance d'une classe, et fournit un point d'accès à cette instance unique.

Contexte Le patron de conception *Singleton* est utilisé lorsqu'un système possède un élément unique (d'une classe particulière), et que l'on veut assurer cette unicité de manière intrinsèque et non simplement par la mise en œuvre du système. Une réponse classique à ce problème est d'utiliser la notion de référence de portée globale, mais cette manière n'empêche pas une deuxième instanciation de la même classe, et de plus peut entraîner des confusions dans les nommages.

De plus, l'utilisation d'une référence de portée globale a un impact fort sur l'architecture du système entier, rendant plus difficile son évolution si l'on désire par la suite autoriser de multiples instances. La mise en œuvre de cette contrainte s'effectuant en un point unique avec le patron *Singleton*, on peut plus facilement faire évoluer le système.

Enfin, la classe *Singleton* permet de sous-classer la classe initiale et de profiter des mécanismes d'héritage.

Structure La classe dont on désire assurer l'unicité définit une méthode de classe renvoyant la référence de l'instance unique, qu'elle peut avoir créé au besoin. Chaque accès à l'instance s'effectuant par le biais de cette méthode de classe, on assure au niveau de la classe l'unicité de l'instance.

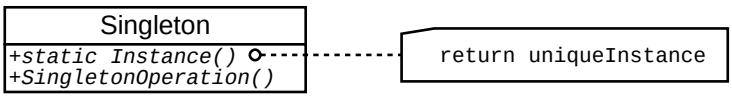

FIG. *3.5* — Patron de conception *Singleton*

Mise en œuvre La mise en œuvre peut varier suivant les capacités du langage utilisé. On utilise le plus souvent une méthode de classe, pour les langages possédant cette notion, qui va intégrer un mécanisme de contrôle (par le biais d'une variable de classe par exemple) sur le nombre d'instances existantes.

Le patron de conception *Chain of Responsibility* (Chaîne de responsabilité)

Objectif Le patron de conception *Chain of Responsibility* encourage le découplage entre une requête et l'objet à laquelle elle est adressée. De cette manière, une même requête peut potentiellement être soumise à différents objets. Une requête peut alors être propagée le long d'une chaîne de différents objets, jusqu'à arriver à l'objet le plus adéquat qui la traitera et ne la transmettra pas plus avant.

Contexte Ce patron est utilisé principalement lorsque l'on ne peut déterminer à l'avance le destinataire final d'une requête, qui peut être potentiellement traitée par plusieurs composants.

Ce sont les composants eux-mêmes qui déterminent s'ils ont la capacité de répondre à la requête, et non le système de plus haut niveau qui leur a transmis le message (par rapport à un patron de conception *Facade* par exemple). L'extensibilité du sysème est alors améliorée : pour traiter de nouvelles requêtes, il suffit d'intégrer dans la chaîne le composant adéquat. Il n'est pas nécessaire d'informer les clients ni une éventuelle *Facade* de la nouvelle fonctionnalité insérée.

Par contre, on ne dispose alors plus de garantie que le message sera traité : puisque l'on ne désigne explicitement aucun destinataire, il se peut que la requête ne soit pas prise en compte. Il faut alors prévoir ce cas dans la chaîne de traitement (par l'insertion d'un composant *par défaut*) ou bien dans les clients.

Structure - Collaboration Le composant *Handler* définit l'interface générique des composants capables de traiter les requêtes. Il peut également fournir le support permettant de chaîner les composants, à savoir une méthode permettant à chaque composant de connaître son successeur.

Chacun des composants particuliers met ensuite en œuvre l'interface spécifiée.

Il suffit alors au client de connaître un point d'entrée dans la chaîne de traitement et de lui envoyer la requête, qui sera alors transmise jusqu'au composant adéquat.

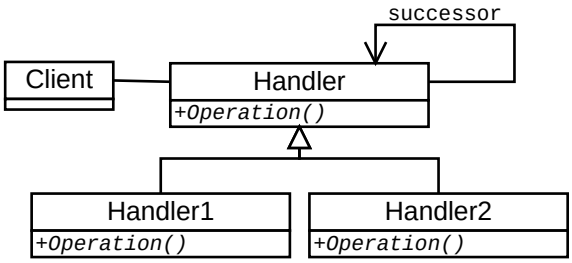

FIG. *3.6* — Patron de conception *Chain of Responsibility*

Mise en œuvre La mise en œuvre impose de se pencher sur deux problèmes : d'une part, la manière de construire la chaîne de composants ; d'autre part, la manière de représenter et de traiter les requêtes.

La chaîne de composants peut être construite de manière répartie, en fournissant à chacun des composants la connaissance de son successeur, ou bien en utilisant les connaissances d'une autre structure, comme par exemple un parent commun à l'ensemble des composants, capable d'énumérer l'ensemble de ses descendants.

La manière de représenter les requêtes dépend du niveau d'extensibilité que l'on veut obtenir. On peut soit utiliser une structure unique, ce qui limitera les possibilités d'extensions, soit utiliser un mécanisme d'héritage. On définit ainsi une classe de haut niveau concentrant les éléments communs à toutes les requêtes, puit on la sous-classe en fonction des besoins plus spécifiques.

3.1.5 Exemples d'applications

Noyau linux

Le noyau linux dispose d'une notion de module [Rhi00], qui représente un ensemble de fonctions et de données que l'on peut insérer ou retirer dynamiquement d'un noyau en cours d'exécution ou lors de son initialisation. Ils sont principalement utilisés pour la définition de pilotes de périphériques. Cette structuration modulaire permet d'utiliser le même noyau linux sur un grand nombre d'architecture, sans avoir à en augmenter exagérément la taille : les modules spécifiques aux composants matériels d'une configuration particulière sont chargés lors de l'initialisation du noyau.

Une des caractéristiques de linux, provenant notamment de son caractère de logiciel libre au développement collaboratif, est d'évoluer rapidement et de présenter parfois, lors des évolutions majeures, une incompatibilité au niveau de son interface de programmation interne. De telles évolutions sont indiquées par un changement de numéro de version majeur ou secondaire.

Ainsi, la transition de la version 2.2 à la version 2.4 a introduit des changements dans l'interface de programmation des modules. Afin de pouvoir concevoir des modules utilisables sur toutes les versions a été développé `kcompat`, une bibliothèque

émulant la nouvelle interface de programmation sur la base de l'ancienne. Les modules développés pour la nouvelle version du noyau peuvent ainsi être utilisés sur une ancienne version, à condition d'utiliser la couche de compatibilité fournie par la bibliothèque.

Par exemple, la fonction `pci_module_init` qui effectue l'intialisation des structures de données associées à un pilote de périphérique connecté sur un bus PCI a été introduite dans le noyau 2.4. Elle est donc présente dans la bibliothèque de compatibilité.

Ce mode de fonctionnement est couramment rencontré dans divers systèmes, particulièrement dans le cas où il est nécessaire d'intégrer des systèmes anciens avec des systèmes plus récents dont l'interface de programmation a changé. Il a donc été identifié dans [GHJV95] comme le patron de conception *Adapter* (Adaptateur).

En continuant l'étude du code du noyau linux, on constate l'utilisation récurrente dans divers sous-systèmes (systèmes de fichiers, couche réseau, etc) d'un type de structure de données rassemblant des références vers un certain nombre de fonctions [BHB99]. Par exemple, la structure `file_operations` regroupe les références vers des fonctions permettant de manipuler des fichiers (ouverture, lecture, écriture, etc). Cette structure constitue une abstraction indépendante du système de fichiers réel (*ext2*, *NTFS*, *NFS*, etc) qui mettra en œuvre les fonctions adéquates pour accéder aux fichiers qu'il contient. Il est ainsi possible d'utiliser la même interface pour accéder à des fichiers se trouvant sur des systèmes de fichiers différents. Cependant, les deux entités (abstraction et mise en œuvre) peuvent évoluer indépendamment.

Ce découplage entre abstraction et mise en œuvre n'est pas propre au noyau linux. On la retrouve dans de nombreux systèmes, en particulier ceux présentant plusieurs alternatives pour accomplir une tâche similaire. Cette structure a donc également été identifiée par les auteurs de [GHJV95] sous le nom de *Bridge* (Pont).

Application Web

Les applications web peuvent fonctionner en utilisant diverses techniques. Parmi celles-ci, on trouve la technologie servlet basée sur le langage java. Le principe de fonctionnement en est le suivant : lorsqu'un navigateur requiert une ressource à un serveur web, ce dernier interroge un *moteur de servlet*. Ce dernier exécute alors une *servlet*, matérialisée par une instance d'une classe Java, qui génère un résultat correspondant à la requête. Ce résultat est ensuite renvoyé au navigateur via le serveur web. La figure 3.7 donne un exemple d'architecture d'un tel système. Dans notre cas, on s'intéresse au fonctionnement d'un service d'annuaire reposant sur l'interrogation d'une base de données ainsi que d'un serveur d'agenda permettant d'obtenir des informations sur les disponibilités de la personne recherchée.

Dans ce système, on identifie différents patrons de conception : la servlet principale permet d'offrir au client une interface unique, que l'on qualifie de *Façade*, pour accéder aux fonctionnalités à la fois de la base de données et du serveur d'agenda.

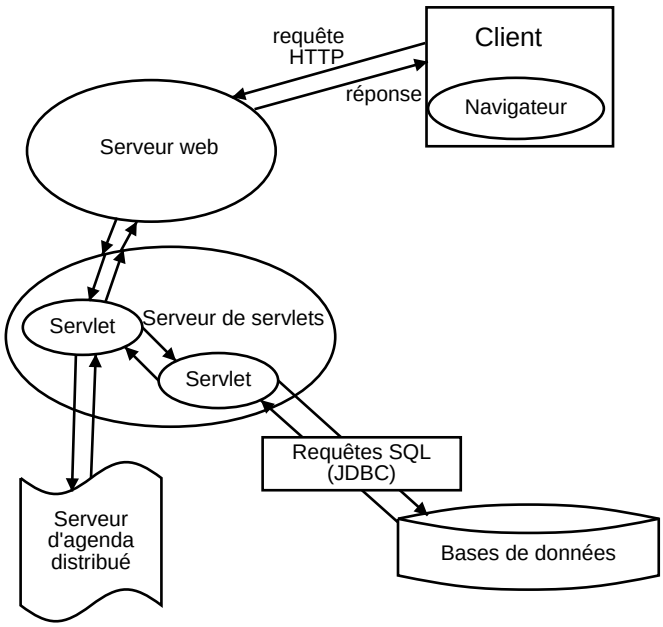

FIG. *3.7* — Principe des servlet

De plus, pour communiquer avec la base de données, la servlet utilise une interface de programmation appelée *JDBC* pour communiquer avec la base de données. Cette interface unique permet, par l'utilisation d'un module approprié, de communiquer avec n'importe quel type de base de données. Ce découplage entre l'abstraction (communication avec une base de données) et la mise en œuvre (communication avec un type de base de données particulier) est un exemple d'utilisation du patron *Bridge* (Pont). En outre, les temps d'établissement de connexion étant souvent importants, la servlet établira une connexion unique qui sera réutilisée pour toutes ses requêtes. Ce point de passage obligé constitue une instance du patron *Singleton*. Enfin, les moteurs de servlet les plus récents fournissent la notion de filtre : il est possible d'appliquer au traitement d'une requête un certain nombre de filtres, qui pourront éventuellement modifier tout ou partie de la requête ou de la réponse. Ce principe de fonctionnement relève du patron de conception *Chain of Responsibility* (Chaîne de responsabilités).

Voyons en détail comment ces différents patrons se retrouvent dans notre architecture, et les conséquences de leur utilisation.

Facade Le patron *Facade* fournit une interface de haut-niveau à un ensemble d'interfaces définies dans des sous-systèmes. Il permet de simplifier l'utilisation du système composé des différents sous-systèmes. Ainsi, les clients n'ont à conserver de référence que sur la *Facade* du système plutôt que sur l'ensemble des sous-systèmes.

La figure 3.4 présente une vue d'ensemble de la mise en œuvre du patron *Facade*.

Le client n'est toutefois pas obligé de passer par la *Facade*, mais peut garder la possibilité de s'adresser directement aux sous-systèmes en fonction de ses besoins.

Dans l'architecture étudiée ici, la servlet principale sert de façade aux deux sous-systèmes que constituent le serveur de base de données et le serveur d'agenda distribué. On pourrait y adjoindre d'autres sous-systèmes tels que l'authentification. Afin d'éviter au client d'avoir à faire plusieurs appels successifs à chacun des sous-systèmes, une interface de consultation unique est proposée.

Bridge Le patron *Bridge* (Pont) vise à découpler, dans un système où plusieurs mises en œuvre du même mécanisme sont possibles, l'interface de la mise en œuvre de l'implémentation. Il est ainsi possible de faire évoluer indépendamment l'un ou l'autre de ces éléments.

La figure 3.3 expose la structure générale du patron *Bridge*. Appliquée au système JDBC qui nous intéresse ici, on constate le découplage entre l'interface proposée par la couche JDBC et les différentes mises en œuvre concrètes proposées pour chacun des types de base de données disponibles. On peut constater des similarités avec le patron *Adapter*, présenté précédemment. La différence repose ici en particulier sur l'intention exprimée à travers l'utilisation de ce patron : le patron *Adapter* permet d'utiliser un système existant en proposant une interface de transition. Le patron *Bridge* est quant à lui utilisé lors de la conception du système, lors de la prise en compte par anticipation de la variété des mises en œuvre possibles.

Singleton Le patron *Singleton* fait qu'un composant ne dispose que d'une instance unique à laquelle tous ses clients s'adressent.

Son utilisation découle de besoins divers :

- pour des raisons de performance (comme dans notre exemple), un point d'entrée unique permet de limiter les coûts d'initialisation ;
- on peut ne disposer que d'une licence unique pour se connecter à une base de données, et l'utilisation du singleton nous en assure ;
- il est nécessaire de maintenir une donnée ou un état global à l'intérieur d'une application.
- etc.

Sa mise en œuvre peut se décliner de plusieurs manières. Il est possible par exemple d'effectuer une mise en œuvre des services uniquement par des méthodes de classe (attribut *static* en java). Une autre manière est d'effectivement créer une instance unique du composant, ce qui laisse des possibilités d'évolution vers une architecture sans singleton par la suite.

Les servlets de manière générale peuvent également être considérées comme une utilisation du patron *Singleton*. En effet, leur fonctionnement classique consiste à créer une seule instance de chaque servlet lors du démarrage du moteur de servlet. Tous les appels aux services qu'elles proposent sont alors effectués sur cette seule

instance.

Chain of Responsibility Le patron *Chain of Responsibility* (Chaîne de responsabilité) évite de lier de manière trop forte l'émetteur d'une requête à son destinataire. Ainsi, il est possible de modifier le destinataire, ou encore de le diviser simplement en un ensemble de fonctions complémentaires qui seront alors invoquées successivement. La notion de *pipe-line* est souvent associée à ce patron de conception.

Comme indiqué dans l'introduction de ce paragraphe, les moteurs de servlet proposent un mécanisme de filtres [Hun01], qui sont des méthodes génériques permettant de modifier tout ou partie de la requête ou de la réponse lors du traitement d'une requête. Ce mécanisme permet de modifier facilement la chaîne de traitement des requêtes, afin d'y ajouter de nouvelles fonctionnalités par exemple. Ainsi, la mise en place d'un contrôle d'accès aux services d'une servlet peut être simplement mise en œuvre, même après le développement de la servlet, par l'introduction d'un filtre d'authentification dans la chaîne de traitement de la servlet en cause. On trouve le même mécanisme à l'œuvre dans la plate-forme de services web Jigsaw [Tea99], dont l'architecture est décrite au paragraphe 5.3.1.

3.2 Motivation

Nous nous proposons de définir ici le patron de conception *Stratégie Autoadaptative* dont l'objectif est de permettre à un système d'acquérir une forme d'autoadaptativité, n'exposant au client qu'une interface unique (*Facade*) accédant à la stratégie la plus adaptée aux conditions d'exécution courantes. Il suffit au client de fournir un accès aux informations d'exécution (environnement) qui permettent de choisir la meilleure stratégie.

Comme nous l'avons vu au chapitre 1, les caches web sont un terrain propice pour la mise en œuvre d'un tel mécanisme. Les différentes politiques (de remplacement, de coopération, etc.) disposent en effet d'un grand nombre de variantes qui sont pour l'instant spécifiées statiquement, bien que les systèmes s'exécutent dans un environnement dynamique. L'adaptation dynamique de ces politiques peut améliorer les performances des caches vis-à-vis d'un objectif de haut niveau (tel que minimiser l'utilisation du réseau par exemple). On trouve le même principe de fonctionnement à l'œuvre dans d'autres applications telles que les agents ou les systèmes mobiles, qui s'inscrivent également dans des environnements dynamiques. L'étude de la mise en œuvre du patron de conception dans des caches web fait l'objet du chapitre 5. Nous allons ici nous intéresser à un exemple tiré d'un système existant dans un autre domaine et proposant déjà un comportement adaptatif, celui des systèmes mobiles, afin de dégager les différents éléments du patron de conception.

Considérons l'exemple d'un poste mobile devant accéder à des données situées sur des serveurs fixes [SKK+90, SA00]. Le dispositif mobile peut être déconnecté du

réseau, et donc incapable d'accéder aux données, ou connecté au réseau et dans ce cas, la qualité de la connexion peut varier de manière importante, offrant un débit variable. Lorsqu'il est connecté avec un débit important (mode *fortement connecté*), le système mobile peut utiliser un mode de transmission standard, comme par exemple un protocole de transfert de fichiers classique. Lors de la déconnexion, une copie des données utilisées doit être créée localement afin que le fonctionnement de l'application ne soit pas perturbé. Un fonctionnement intermédiaire (mode *faiblement connecté*), en présence de perturbations, peut amener à dégrader le comportement de certaines fonctionnalités. On observe dans notre exemple trois comportements distincts : le mode fortement connecté, le mode faiblement connecté et le mode déconnecté. Le système mobile doit déterminer le mode de fonctionnalité courant, en fondant sa décision sur l'obtention d'informations de son environnement important. Deux aspects interviennent dans ce problème : l'aspect fonctionnel (stockage des données) et l'aspect d'adaptation (détermination du mode de fonctionnement et choix du mécanisme adapté). Nous souhaitons offrir de la manière la plus simple possible une résolution des deux aspects.

3.3 Structure du patron de conception

L'étude des systèmes adaptatifs présentée dans le chapitre 2 nous a permis de dégager quatre grandes étapes dans le processus d'adaptation. Tout d'abord, le système doit être capable d'**obtenir des informations sur son environnement important**. Ceci peut être effectué soit par observation, soit par un mécanisme de notification. À partir des informations recueillies, le système **prend une décision** et sélectionne, si besoin est, un nouveau comportement. S'il décide de remplacer le comportement courant, il peut avoir à **gérer la transition** entre l'ancien et le nouveau comportement. Cette phase implique typiquement la transmission d'informations sur l'état des comportements. Dans notre exemple, ce peut être la liste des messages en attente : lors du passage d'un mode déconnecté à un mode connecté, la liste des messages en attente d'émission doit être transmise de la stratégie de communication déconnectée à la stratégie de communication connectée [SA00]. Une fois que le nouveau comportement est prêt à être activé, le système peut enfin **valider les changements** et activer le nouveau comportement. Le système est alors prêt à répondre à de nouveaux changements. La figure 3.8 résume le cycle que nous venons de présenter.

Le patron de conception *Strategy* [GHJV95] suggère déjà une manière de mettre en œuvre un choix dynamique entre plusieurs stratégies, par le biais de leur réification au sein de composants respectant une interface commune. Nous allons donc utiliser ce patron de conception, comme présenté dans la figure 3.9, pour mettre à la disposition du client les différentes stratégies possibles.

Cependant, le choix du comportement adéquat est laissé au client, et les mécanismes permettant d'effectuer ce choix ne sont pas évoqués. Le patron de conception

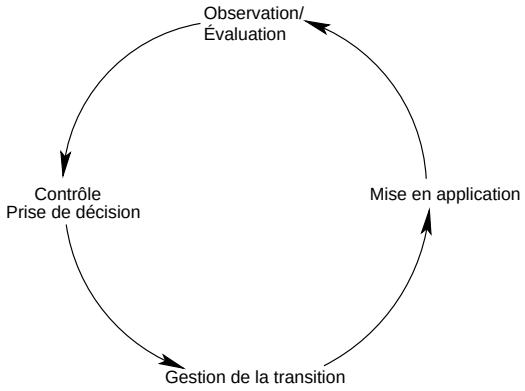

FIG. 3.8 — Le cycle de l'adaptation

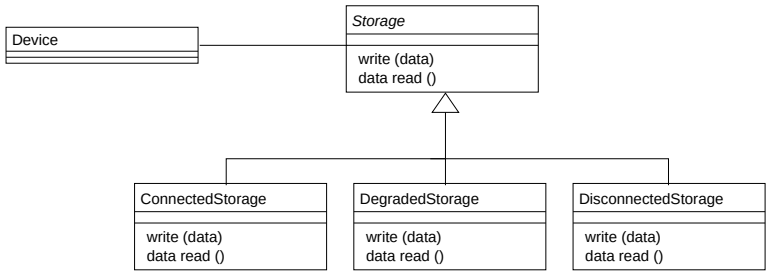

FIG. 3.9 — Application du patron de conception *Strategy*

Strategy est donc nécessaire, mais pas suffisant, pour traiter le processus d'autoadaptation. Dans notre description du processus d'adaptation, nous pouvons trouver une étape d'observation/introspection. Le patron de conception *Observer* offre une méthode de résolution de ce type de problème. Il est possible de le mettre en œuvre entre un module de gestion du réseau (NetworkMonitor dans le rôle du sujet) et le client (dans le rôle de l'observateur), comme le présente la figure 3.10. Toutefois, on entretient encore avec ce type d'architecture une confusion entre le processus d'exécution (le stockage) et le processus d'adaptation (le contrôle). Le client n'a pas forcément à être informé des changements d'état du réseau. Le principe de séparation des aspects (*separation of concerns*) nous conduit donc à introduire un composant Controller, nouvel observateur du NetworkMonitor, chargé de recevoir les informations relatives à la qualité de la connexion. La figure 3.11 présente la nouvelle architecture relative à la phase d'observation/introspection.

Nous disposons à présent de deux sous-systèmes : un relatif à la mise en œuvre des différentes stratégies, et un autre dédié à l'observation des conditions de fonctionnement. Afin de faciliter l'utilisation de ces sous-systèmes par le client, nous utilisons le patron de conception *Facade*, et introduisons un composant AdaptiveStorage présentant une interface unifiée au client qui peut la considérer comme une stratégie comme une autre, d'où sa position dans le patron *Stratégie Autoadaptative*.

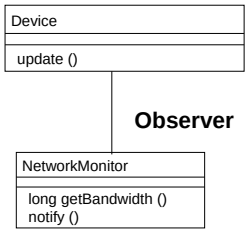

FIG. *3.10* — Application directe du patron de conception *Observer*

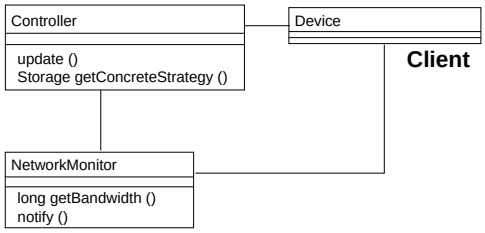

FIG. *3.11* — Introduction du Controller

La figure 3.12 montre l'articulation de ces différents systèmes. Le périphérique Device utilise pour stocker ses données les services de la stratégie Storage par le biais du composant AdaptiveStorage . La stratégie abstraite Storage se décline en plusieurs stratégies concrètes ConnectedStorage, DisconnectedStorage et DegradedStorage. Le choix de la stratégie concrète pertinente est effectué par le composant Controller en fonction des informations qui lui sont fournies par le composant NetworkMonitor. Lors du passage du mode déconnecté au mode connecté, la liste des données à stocker, mémorisée par la stratégie concrète DisconnectedStorage, est transférée à la nouvelle stratégie ConnectedStorage par le composant StateAdapter.

Nous pouvons traduire ce modèle en un patron de conception plus général, dont la structure est représentée dans la figure 3.13, rendant explicites les différents éléments nécessaires au système pour réaliser l'adaptation :

- la passerelle d'informations, appelée InformationGateway, offre les fonctionnalités de supervision (*monitoring*) et d'observation ;
- le contrôleur, appelé Controller, est chargé de choisir le nouveau comportement ;
- le composant d'adaptation, appelé StateAdapter, est dédié au processus d'adaptation et de transmission des informations entre les stratégies lors de la phase de transition ;
- le composant AdaptiveStrategy constitue le point d'entrée pour les clients.

3.3.1 Participants

Les composants du patron *Stratégie Autoadaptative* sont organisés suivant d'autres patrons de conception, à savoir *Strategy*, *Facade* et *Observer*. Nous détaillons ici le rôle de chacun des composants représenté sur la figure 3.13.

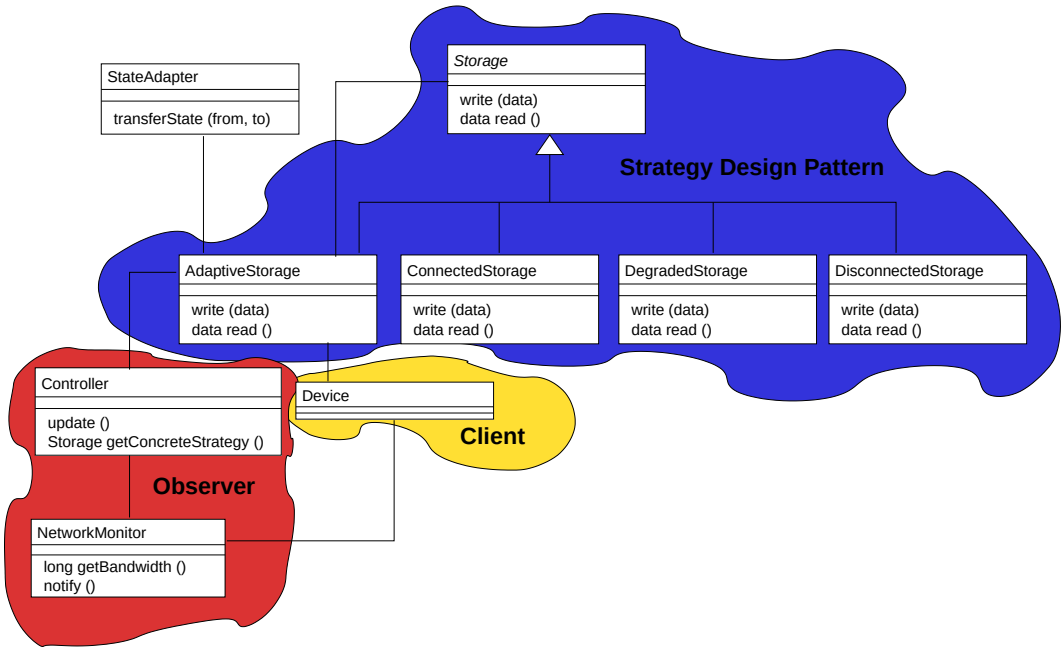

FIG. *3.12* — Exemple de modèle de classes

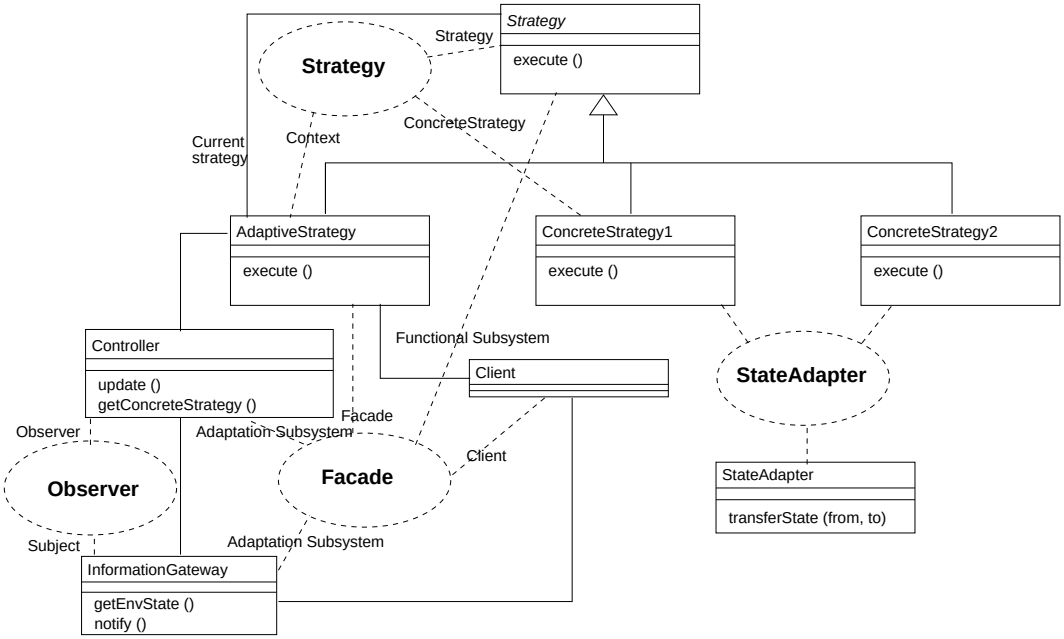

FIG. *3.13* — Structure générale du patron de conception

Client accède aux services de la stratégie via le composant AdaptiveStrategy, et fournit des informations sur ses conditions de fonctionnement via le composant InformationGateway.

Strategy déclare une interface commune aux différentes stratégies concrètes utilisables. Il représente la stratégie abstraite. Client utilise cette interface pour invoquer le service défini par un composant ConcreteStrategy.

ConcreteStrategy implémente une stratégie concrète, en respectant l'interface définie par Strategy. Chaque ConcreteStrategy peut comprendre notamment un état, selon la nature du service implémenté.

AdaptiveStrategy est le principal point d'accès pour les clients du système adaptatif, et fonctionne comme une *Facade* du système. Il fournit un point d'accès unique aux fonctionnalités des différentes stratégies.

Un aspect essentiel de notre patron de conception est le fait que l'objet AdaptiveStrategy hérite lui-même de Strategy. En effet, ceci contribue à simplifier le modèle, et rend plus explicite le découplage entre le mécanisme implémenté par le composant Strategy et le mécanisme d'adaptation. On peut ainsi envisager de remplacer de manière transparente le composant AdaptiveStrategy par un des composants ConcreteStrategy, dans le cadre d'une optimisation par spécialisation par exemple.

Controller doit effectuer le choix de la meilleure stratégie, en utilisant l'information fournie par InformationGateway, mettant en œuvre le patron de conception *Observer*. Il est invoqué par le composant AdaptiveStrategy.

InformationGateway représente le canal par lequel le composant Controller obtient les informations nécessaires à sa prise de décision. Il fournit des méthodes d'accès aux paramètres d'environnement.

- Il définit une interface qui permet au Controller d'accéder aux données courantes ou historiques de l'environnement d'exécution.
- Il peut également proposer des fonctionnalités actives, déclenchant des actions en cas de changement dans l'environnement, via des notifications aux autres composants. Cette fonctionnalité peut utiliser le patron de conception Observer.

StateAdapter gère si besoin est les transitions entre deux stratégies dans le cas où il est nécessaire d'échanger, en les transformant éventuellement, des informations sur leurs états respectifs. Nous faisons apparaître dans le schéma un proto-patron de conception *StateAdapter*. Celui-ci n'a pas encore fait l'objet description poussée, mais lors de notre étude, nous avons constaté qu'il apparaissait dans de nombreux systèmes. Les patrons de conception *State* et *Memento* peuvent définir des mécanismes de base utilisés à cette fin, comme nous le voyons dans le paragraphe 3.5.3, mais les enjeux de la phase de transition ne sont pas examinés en détail.

3.3.2 Collaborations

On peut distinguer deux types d'adaptation [BMN+00] : l'adaptation *on action* et l'adaptation *on change*. L'adaptation *on action* est effectuée lors de l'invocation d'une méthode de la stratégie. Ce type d'adaptation est utilisé lorsque les paramètres utilisés pour choisir la stratégie concrète font figurer des paramètres de calcul. L'adaptation *on change* est déclenchée par un changement dans l'environnement, typiquement relevé par une notification du composant InformationGateway, indépendamment des invocations des méthodes. L'adaptation effectuée par le système mobile présenté au paragraphe 3.2 est un exemple d'adaptation *on change*.

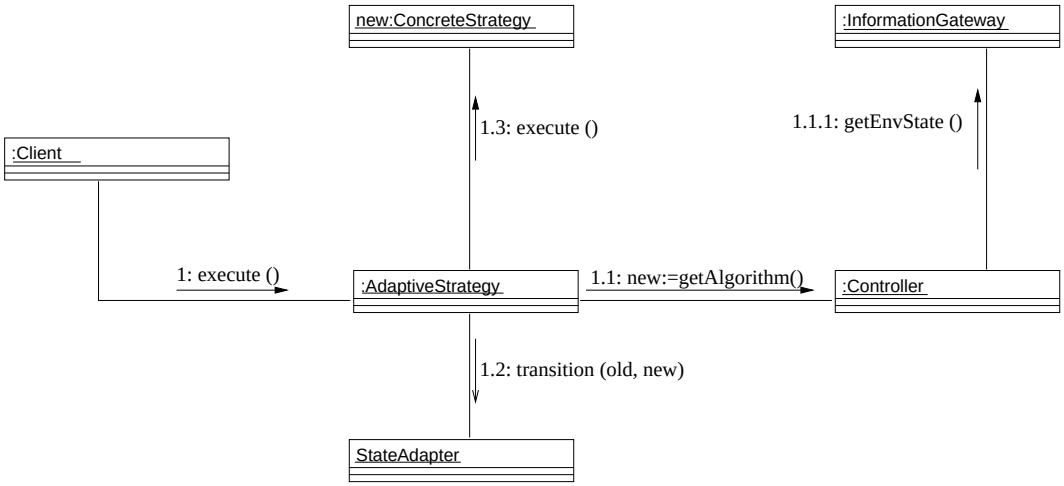

FIG. *3.14* — Diagramme de collaboration pour une adaptation *on action*

La figure 3.14 donne un exemple de collaboration entre les différents éléments de notre patron de conception dans le cas d'une adaptation *on action*. Le composant Adaptative Strategy constitue le point d'entrée principal pour les clients (le composant Context dans le patron de conception original *Strategy* [GHJV95]). Il reçoit une requête (*1*) pour une opération, et s'enquiert (*1.1*) auprès du Controller du composant ConcreteStrategy la plus adaptée pour effectuer l'opération dans le contexte courant. Le Controller choisit la meilleure stratégie, en se fondant sur les informations fournies (*1.1.1*) par l'InformationGateway, et informe le composant Adaptive Strategy de son choix. S'il apparaît qu'un changement de stratégie est nécessaire, et que l'ancienne et la nouvelle stratégie partagent une information d'état, le composant StateAdapter est mis à contribution pour gérer l'adaptation et le transfert (*1.2*) des informations pertinentes de l'ancienne stratégie vers la nouvelle. Une fois cette transition effectuée, le composant Adaptive Strategy peut enfin transmettre la requête (*1.3*) au composant ConcreteStrategy appropriée.

La figure 3.15 présente une collaboration entre les différents éléments de notre patron de conception dans le cas d'une adaptation *on change*. Dans ce cas de figure,

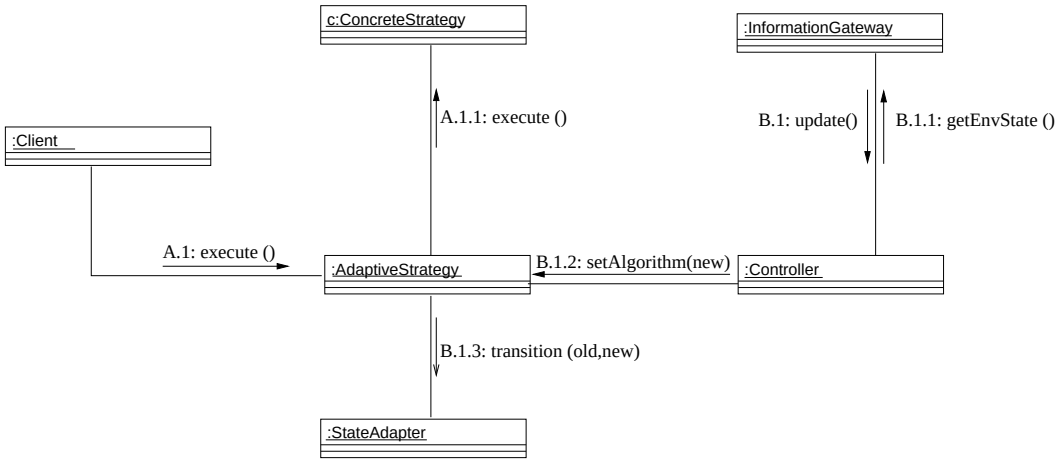

FIG. *3.15* — Diagramme de collaboration pour une adaptation *on change*

le *traitement de la requête*, indexé par la lettre A, est indépendant du *processus d'adaptation*, indexé par la lettre B.

Durant le processus d'adaptation, le composant InformationGateway indique (*B.1*) au Controller, par le biais d'une notification, que les paramètres d'environnement ont changé. Le Controller s'enquiert (*B.1.1*) alors des nouvelles valeurs des paramètres et décide sur cette base si la stratégie courante doit être remplacée. Si un changement est nécessaire, le Controller indique (*B.1.2*) au composant Adaptative Strategy la nouvelle ConcreteStrategy devant être activée. Le composant Adaptative Strategy peut si nécessaire être amené à invoquer (*B.1.3*) le StateAdapter pour adapter et transférer des informations entre l'ancienne et la nouvelle stratégie.

Indépendamment du processus d'adaptation, les requêtes (*A.1*) sont traitées par le composant AdaptiveStrategy, qui transmet (*A.1.1*) directement les requêtes à la ConcreteStrategy courante. Le cas de requêtes arrivant au moment de la transition entre deux stratégies doit bien sûr être traité par le composant AdaptiveStrategy.

3.4 Conséquences

L'utilisation du patron de conception *Stratégie Autoadaptative* amène les propriétés suivantes :

- *Il masque des détails au client. Stratégie Autoadaptative* fournit une interface unique permettant d'invoquer le meilleur algorithme, sans que le client ait besoin de connaître toutes les alternatives ni la méthode de choix. Il tire parti d'une connaissance des conditions d'exécution lui permettant de choisir la stratégie la plus appropriée. Le client doit néanmoins fournir un certain nombre d'informations, identifiées par le biais du composant InformationGateway.

- *Il distingue le processus d'adaptation du processus d'exécution. Stratégie Autoadaptative* favorise une séparation plus nette entre ces deux aspects, conduisant à une plus grande clarté dans la compréhension des mécanismes et de leurs interactions.
- *Il rend la phase de transition explicite.* La phase de transition est souvent négligée dans les systèmes adaptatifs. Le composant StateAdapter réifie cet aspect important.
- *Il peut s'appliquer de manière récursive.* Il est possible d'appliquer le patron de conception *Stratégie Autoadaptative* à ses propres composants afin de profiter de ses propriétés à différents niveaux (observation, contrôle), comme décrit dans le paragraphe 3.7.
- *Il a un impact sur l'extensibilité.* L'addition d'une nouvelle stratégie est simplifiée par la réification effectuée. Cependant, le Controller doit connaître l'ensemble des stratégies disponibles ainsi que leurs caractéristiques respectives, afin de pouvoir choisir la plus appropriée. Le choix d'implémentation du mécanisme du Controller, abordée dans le paragraphe 3.5.2, est donc crucial au regard de l'extensibilité.
- *Il a un impact sur les performances.* La mise en œuvre du patron *Stratégie Autoadaptative* peut entraîner des surcoûts dûs aux mécanismes utilisés. Ainsi, les indirections supplémentaires au niveau du code doivent être prises en compte dans l'évaluation du coût du processus d'adaptation. Les phases d'observation (paragraphe 3.5.1), de prise de décision (paragraphe 3.5.2) et de transition (paragraphe 3.5.3) ont également une incidence sur les performances du système.

3.5 Mise en œuvre

Au paragraphe 3.2, nous avons identifié quatre étapes pour le processus d'adaptation. Nous pouvons, comme représenté dans la figure 3.16 associer un composant de notre patron de conception à chacune de ces étapes.

- évaluation et surveillance des paramètres d'exécution (InformationGateway) ;
- prise de décision (Controller) ;
- gestion des transitions (StateAdapter) ;
- validation des changements (AdaptiveStrategy).

Nous allons reconsidérer chacune de ces étapes afin d'étudier les différentes options présentes lors de leur mise en œuvre.

3.5.1 L'évaluation des paramètres

L'évaluation et la surveillance des paramètres est un point essentiel d'un système autoadaptatif. Les points observés doivent être pertinents au regard du mécanisme de prise de décision. De plus, le coût de l'observation est à prendre en compte dans le coût total de l'adaptation [OGT+99] car une approche trop intrusive ou extensive

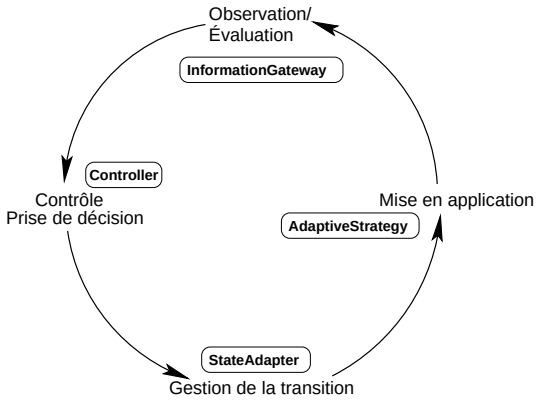

FIG. 3.16 — Le cycle de l'autoadaptativité et les composants du patron de conception *Stratégie Autoadaptative*

peut entraîner une dégradation des performances.

L'observation du contexte important peut se faire de manière passive ou active. En d'autres termes, le Controller peut avoir besoin de connaître certains paramètres lors de sa prise de décision. Le composant InformationGateway est alors passif, ne servant que d'intermédiaire pour transmettre les valeurs observées. Une approche plus active du composant InformationGateway peut être nécessaire. D'une part, certains éléments d'information peuvent nécessiter une mise à jour continuelle, comme par exemple l'accumulation d'informations d'historiques. Le composant InformationGateway doit donc être capable de gérer le flux de données à sa disposition, pour générer ensuite, lors des sollicitations du Controller, des informations plus synthétiques. D'autre part, le processus d'adaptation *on change* implique forcément un comportement actif de la part du composant InformationGateway, qui est chargé de la notification du Controller en cas de changement notable dans l'état du système.

Par ailleurs, nous précisions dans notre étude de l'adaptation au chapitre 2 que la temporalité était une propriété fondamentale des systèmes adaptatifs. Elle est traduite dans le composant InformationGateway par la possibilité de donner accès à trois types d'informations, concernant le passé, le présent et le futur des paramètres.

Le passé concerne les informations historiques que le système conserve afin de pouvoir affiner ses choix. Les systèmes informatiques génèrent souvent de nombreuses informations d'historique, qui sont souvent sous-exploitées. Ainsi, dans un cache web, on peut trouver une trace des requêtes effectuées.

Le présent concerne les paramètres courants, mesurables instantanément tels que la bande passante d'un système en réseau, la quantité de mémoire disponible, etc., ou bien synthétisant un état du système comme par exemple le nombre de requêtes traitées depuis la mise en marche du système.

Le futur concerne les informations prospectives qu'il est possible de générer à partir des deux précédentes. Les systèmes de préchargement de données de caches

web, décrits dans le paragraphe 1.7.3, peuvent tirer profit d'une prévision des accès futurs à des documents ou de l'utilisation de bande passante du système. De même, Narayanan *et al.* [NFS00] ont étendu le système *Odyssey* afin d'y inclure un système de prédiction basé sur l'historique de fonctionnement, qui peut estimer la consommation de ressources d'une application en fonction de son mode de fonctionnement.

On retrouve notamment cette distinction passé/présent/futur dans le modèle d'environnement donné dans le *Viable System Model* de Beer, introduit dans le paragraphe 2.2.1, ainsi que dans le système de *monitoring* du système d'exploitation adaptatif VINO [SS98]. Ce dernier collecte un historique des événements survenus dans le système (informations passées), permet d'accéder en temps réel à des informations sur l'état du système (informations présentes), et, par l'utilisation de simulations *in-situ*, contribue à la prévision du fonctionnement futur du système dans les divers scénarios envisagés.

En ce qui concerne la structuration du système d'observation des systèmes autoadaptatifs, elle est généralement hiérarchique [DB00, SA00]. Le système d'observation de MolèNE constitue un exemple typique de cette architecture. MolèNE utilise pour l'interaction avec le contexte un système de détection et de notification de variations, appelé SDN, représenté dans la figure 3.17. On voit apparaître dans son appellation les deux aspects d'un InformationGateway précisés précédemment, qui sont la consultation directe des paramètres et la notification des changements.

Le SDN adopte une architecture hiérarchique, reposant sur des *moniteurs de base* qui effectuent des mesures basiques des paramètres du système (moniteur de bande passante, moniteur de mémoire, moniteur de batterie). Ces composants ont une approche active qui leur permet d'informer les composants de plus haut niveau en cas de changement notable d'un paramètre.

Les composants de plus haut niveau sont appelés *moniteurs de haut niveau* (MHN), et effectuent la synthèse des informations reçues par les moniteurs de base. Ce sont les points d'entrée du SDN, et ils proposent également soit une consultation directe des paramètres, soit un mécanisme de notification programmable.

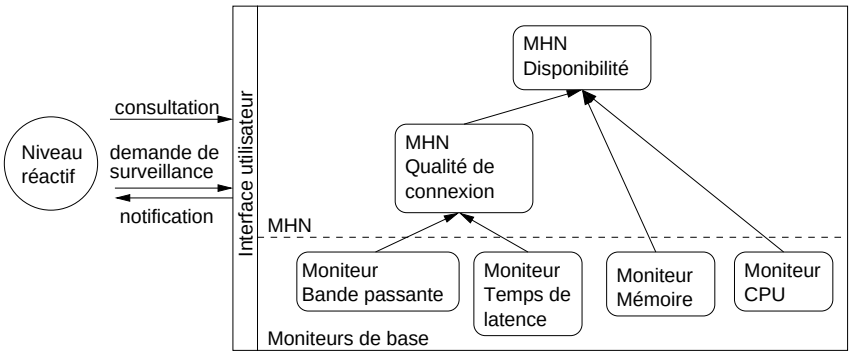

FIG. 3.17 — Le système de détection et de notification de variations de MolèNE

3.5.2 La prise de décision

Le Controller est la *concrétisation de l'objectif* du système adaptatif. Il implémente la *politique d'adaptation* du système, ou dans les termes utilisés par Holland et repris dans le paragraphe 2.6.4, le *plan d'adaptation*. Il peut lui-même être paramétré ou non : il peut représenter l'objectif à atteindre de manière absolue, ou bien accepter en paramètre un facteur de qualité par exemple.

Le Controller est chargé de choisir l'algorithme le plus adapté vis-à-vis des conditions d'exécution. Diverses approches peuvent être mises en œuvre sur cet aspect, et les plus courants dans les systèmes existants utilisent des fonctions de coût, des automates d'états, ou encore des simulateurs.

L'utilisation de fonctions de coût est couramment appliquée dans des objectifs d'optimisation de paramètres d'un algorithme. Si l'on considère les différentes stratégies possibles comme autant de valeurs pour un paramètre du système, on peut appliquer cette méthode simple de sélection. La difficulté réside dans l'établissement des fonctions de coût associées à chaque stratégie concrète, ce qui explique peut-être que cette voie soit relativement inexplorée dans le domaine de l'adaptation de comportements.

Le système MolèNE [SA00] utilise au sein de son entité réactive un *automate* dont les états correspondent à des conditions d'exécution et les transitions aux actions à entreprendre lorsqu'une variation sur un paramètre implique une adaptation. Cette approche assez générale permet une grande flexibilité, et facilite la conception de diverses stratégies d'adaptation.

Il est enfin possible d'utiliser un *simulateur* afin d'évaluer différents scénarios possibles. Ainsi, le système d'exploitation VINO [SS98] permet d'utiliser la plupart de ses modules système dans un mode de simulation, ne modifiant pas l'état global du système. Cette approche permet d'offrir une bonne base à la simulation *in situ* des différentes possibilités d'évolution du système.

3.5.3 La phase de transition

La phase de transition comporte deux aspects importants. D'une part, un mécanisme doit être mis en place pour contrôler l'activation de la nouvelle stratégie et la désactivation de l'ancienne. D'autre part, il peut être nécessaire de transférer des informations de l'ancienne stratégie vers la nouvelle.

L'introduction de ces deux mécanismes entraîne un coût de transition, qu'il faut prendre en compte lorsque l'on évalue le gain espéré avant de décider de changer de comportement [LLB97, KC99]

Le mécanisme

Divers mécanismes peuvent être utilisés dans cette phase de transition. Ils dépendent des contraintes appliquées au système, ainsi que des propriétés des stratégies considérées. Leur mise en œuvre peut être effectuée au niveau du composant AdaptiveStrategy.

L'approche la plus simple consiste à geler le système durant la phase de transition. On procède alors à l'arrêt de l'ancienne stratégie et à l'activation de la nouvelle. C'est une méthode utilisée par exemple dans les systèmes de migration de processus [MW00, Pla97]. Cependant, elle peut être incompatible avec les exigences du système.

Si les deux stratégies peuvent fonctionner en parallèle, une désactivation progressive peut être entreprise. Nous avons par exemple implémenté une stratégie de remplacement de données adaptative dans un simulateur de cache web, dont nous parlerons dans le paragraphe 5.2.2. Pour cette stratégie, nous avons introduit une stratégie, appelée ReplSwitch, chargée de répartir les requêtes durant la phase de transition. Le même mécanisme est utilisé dans le système 2K [KC99] : l'ancienne stratégie peut continuer de fonctionner tant que des clients l'utilisent, mais les nouvelles requêtes sont transmises à la nouvelle stratégie.

La transmission d'informations d'état

Selon l'implémentation des stratégies, la phase de transition d'une stratégie à l'autre peut entraîner le transfert d'informations sur l'état de l'ancienne stratégie vers la nouvelle. On peut distinguer les stratégies *sans état* des stratégies *avec état*. Le caractère sans état d'une stratégie entraîne des simplifications dans le modèle d'autoadaptation. Par exemple, des stratégies de tri ne partagent pas d'information d'état. Par contre, une stratégie de gestion de la concurrence [KC99], proposant comme alternatives les stratégies concrètes ThreadPool, Single-Threaded et Thread-Per-Connection, peut avoir besoin de transmettre d'une stratégie à l'autre la liste des processus activables. Le composant StateAdapter est chargé de ce transfert, qui peut impliquer une adaptation des données.

Quelques mécanismes ont déjà été développés pour des applications spécifiques. Le *checkpointing* [Pla97] ou les mécanismes de persistance [Per] ont été employés dans le domaine de la migration de processus[MW00] par exemple, pour permettre de capturer l'état d'un système et sa restauration ultérieure. Cependant, ces mécanismes sont orientés vers la restauration d'état dans un système inchangé, et non vers un nouveau système. Dans cette perspective, Hicks [Hic01, HMN01] identifie trois tâches qu'un système de transfert d'état doit effectuer :

- identifier l'état qui devra être transféré vers le système de destination. Cet état est appelé *état persistent*. Les autres informations sont qualifiées d'*éphémères* ;
- développer un mécanisme d'encodage et de décodage de l'information, et un moyen de la transférer du système original au système de destination ;

– intégrer dans le système de destination un moyen d'intégrer un état ainsi transmis.

Ainsi, dans un cache web implémentant diverses stratégies de remplacement (voir le paragraphe 1.7.7), une des informations essentielles des stratégies consiste souvent en une liste ordonnée des documents présents dans le cache. Cette liste est ordonnée suivant des critères dépendant de la stratégie utilisée, et représente une vue particulière sur une donnée interne du système (la liste des documents stockés, avec diverses méta-informations). Si on considère que chaque instance de stratégie contient une telle liste, il devient nécessaire, lors du passage d'une stratégie à une autre, d'initialiser les données de la nouvelle stratégie avec les informations présentes dans l'ancienne.

MolèNE [SA00] fournit un service nommé state adapter qui permet de transférer des informations entre les différentes stratégies nommées Implementation. Si l'information est suffisamment similaire, le service relève du patron de conception *Memento*. Si l'information se présente sous des formes différentes, le service de *state adapter* doit sélectionner l'information pertinente de l'Implementation source afin de l'adapter aux besoins de la nouvelle Implementation.

Le système d'exploitation distribué 2K [HKC+99] utilise un ORB CORBA personnalisé, appelé dynamicTAO [RKC99]. Ce dernier gère les transitions via un composant dédié appelé *ComponentConfigurator* [KC99], qui utilise également un patron de conception *Memento*.

3.5.4 La validation

Une fois que la nouvelle stratégie a été choisie et que la transition éventuelle d'informations d'état a été effectuée, le composant AdaptiveStrategy peut valider l'adaptation et activer la nouvelle stratégie. La manière dont cette validation est mise en œuvre dépend du mécanisme utilisé dans la phase de transition.

3.6 Combinaison des types d'adaptation

Comme précisé dans le paragraphe 3.3.2, [BMN+00] identifie deux types d'adaptation : l'adaptation *on change* qui est déclenchée par un changement notable dans l'environnement d'exécution, et l'adaptation *on action* qui a lieu lors de l'invocation d'une stratégie.

Ces deux types d'adaptation peuvent être combinés suivant le comportement désiré de l'application. On peut par exemple envisager une stratégie d'affichage dans un environnement mobile proposant une adaptation *on change* suivant le mode de connexion, comme présenté dans le paragraphe 3.2, et une adaptation *on action* suivant le type des données affichées. Deux approches sont envisageables pour mettre en place cette combinaison. Il est possible d'implémenter les différents mécanismes

relevant de l'adaptation *on change* et *on action* dans le Controller. Ce dernier est alors activé, d'une part par le composant InformationGateway dans le cas de l'adaptation *on change*, et d'autre part par le client à chaque invocation de méthode dans le cas de l'adaptation *on action*. Cependant, cette implémentation peut compliquer le code du Controller.

La seconde approche consiste à appliquer le patron de conception *Stratégie Autoadaptative* au Controller lui-même, ce qui revient à une application récursive du patron de conception, mentionnée dans le paragraphe 3.7. En effet, ce dernier peut être conçu comme une stratégie autoadaptative, qui change de stratégie de sélection en fonction de ses conditions d'exécution. L'adaptation *on change* s'applique alors au Controller, qui joue ici le rôle de stratégie abstraite, et les différentes stratégies concrètes de contrôle sont alors utilisées pour le contrôle effectif de l'adaptation *on action*.

Si l'on reprend l'exemple du système d'affichage, une stratégie abstraite Display est implémentée par plusieurs stratégies concrètes DisplayText-*, DisplayPicture-*, DisplayMovie-*, etc., déclinées chacune en trois versions suivant le mode de connexion , que nous suffixons par *-d* pour déconnecté, *-l* pour faiblement connecté et *-h* pour fortement connecté).

La première approche, dont le principe est présenté dans la figure 3.18, consiste à introduire dans le même Controller les mécanismes permettant d'effectuer l'adaptation *on change* lors de la notification d'un changement de mode de connexion, et l'adaptation *on action* lors de l'activation de la stratégie.

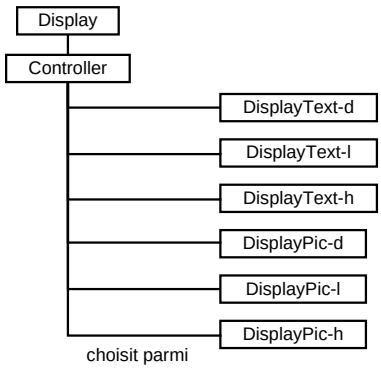

FIG. *3.18* — Combinaison d'adaptations - première variante

La seconde approche, appliquant de manière récursive le patron de conception, peut se décliner en deux variantes suivant la priorité que l'on veut accorder à l'une ou l'autre des deux adaptations, mais également suivant des critères de facilité de mise en œuvre.

On peut, comme présenté dans la figure 3.19, isoler l'adaptation *on change* dans un contrôleur dédié à chaque type de média. Ainsi, on crée les contrôleurs Controller-Text, Controller-Pic, etc. qui sont sélectionnés dans Controller par le biais d'une adap-

tation *on action*. L'adaptation *on change* est gérée au niveau de chaque variante du Controller.

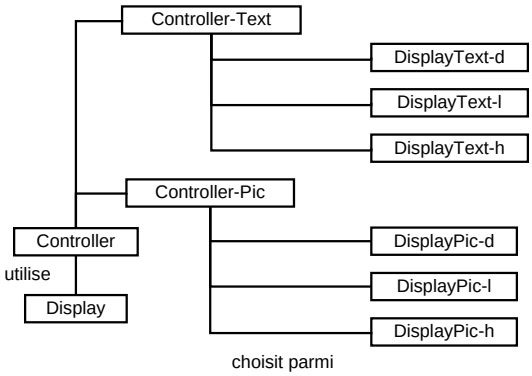

FIG. 3.19 — Combinaison d'adaptations - deuxième variante

La deuxième variante, présentée dans la figure 3.20, consiste à mettre l'accent sur l'adaptation *on change*, qui permet de déterminer le mode de fonctionnement en fonction de la bande passante mesurée. On définit ainsi trois Controller spécifiques à chaque mode de connexion, qui effectuent uniquement une adaptation *on action* : Controller-d, Controller-l et Controller-h. Chacun des trois Controller-* ne s'intéresse qu'aux stratégies d'affichage du mode approprié. Le mécanisme d'adaptation *on change* est mis en œuvre dans un MetaController.

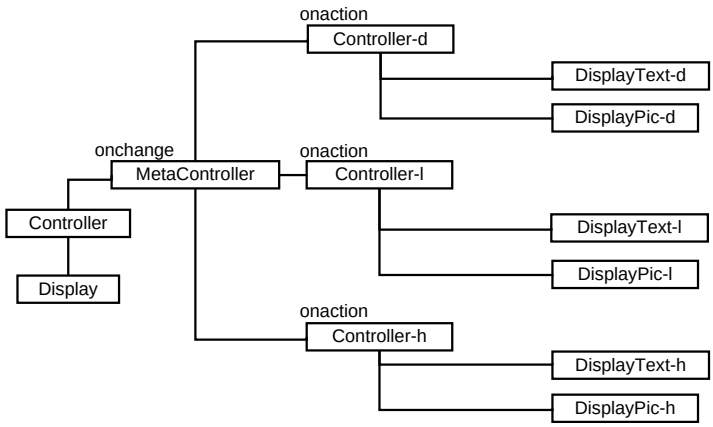

FIG. 3.20 — Combinaison d'adaptations - troisième variante

La première approche ne tire que peu parti des propriétés du patron de conception *Stratégie Autoadaptative*, notamment de la séparation des aspects (*separation of concerns*). Nous nous intéressons donc plus particulièrement aux deux variantes de la seconde approche. Les deux variantes appliquent le patron de conception *Stratégie Autoadaptative* de manière récursive, transformant le Controller utilisé par la stratégie Display en une *Stratégie Autoadaptative*. Cependant le choix de l'une ou

l'autre des variantes n'est pas sans conséquences en particulier sur l'extensibilité du système, comme on le mentionnait au paragraphe 3.4, et sur ses performances.

En effet, l'extensibilité du système, c'est-à-dire sa capacité à prendre en compte de nouveaux besoins, dépend du choix de mise en œuvre de l'adaptation au niveau du Controller. Dans la première variante, il suffit en cas d'ajout de nouveaux médias de créer un nouveau contrôleur spécifique au média introduit, et de l'enregistrer auprès du Controller global. Dans la deuxième variante, il est nécessaire de mettre à jour chacun des contrôleurs *on action*. La même remarque s'applique de façon symétrique si l'on envisage l'ajout d'un nouveau mode de fonctionnement.

Les performances du système peuvent être affectées par le mécanisme de notification de l'adaptation *on change*. Ce dernier peut être moins efficace s'il doit diffuser l'information à plusieurs contrôleurs distincts plutôt qu'à un seul contrôleur. La seconde variante semble donc ici plus pertinente au regard de ce paramètre.

3.7 Récursivité d'application

Ce patron de conception peut être appliqué sur lui-même de manière récursive. On peut l'utiliser dans le Controller afin de fournir un contrôleur adaptatif, pour implémenter à la fois une adaptation *on change* et une adaptation *on action*, comme décrit dans le paragraphe 3.6.

On peut l'utiliser également dans le InformationGateway, afin d'adapter l'observation aux conditions courantes, prenant alors en compte le Controller et la stratégie en activité. Dans l'exemple proposé dans le paragraphe 3.5.3, on peut alors obtenir une autre manière d'adapter la liste des documents à la politique courante, en intégrant cet aspect directement dans le composant InformationGateway.

Cette utilisation illustre notamment l'aspect récursif présenté dans le Viable System Model, présenté dans le paragraphe 2.2.1.

3.8 Conclusion

Le patron de conception *Strategy* présente un mécanisme permettant d'effectuer dynamiquement un choix entre diverses variantes d'une stratégie. Cependant, il n'offre – à juste titre – pas d'indications particulières sur la manière d'effectuer ce choix. Dans le cadre des systèmes autoadaptatifs, nous désirons augmenter l'autonomie des systèmes informatiques. Parmi les techniques proposant déjà ce type d'approche, nous avons pu identifier des aspects communs, ce qui nous a permis d'identifier un processus d'adaptation et de dégager un patron de conception *Stratégie Autoadaptative* facilitant la mise en œuvre de ce processus. Ce patron de conception insiste particulièrement sur la notion de séparation des aspects (*separation of concerns*) entre le processus d'exécution et le processus d'adaptation. Il peut de plus intervenir à différents niveaux du système concerné, et s'appliquer aussi bien au pro-

cessus d'exécution qu'au composant d'observation ou à celui de contrôle. Il répond en outre aux besoins des deux types d'adaptation identifiés : l'adaptation *on change* et l'adaptation *on action*.

Nous allons voir dans le chapitre 4 comment on peut utiliser le patron de conception présenté pour analyser des systèmes autoadaptatifs, ainsi que la manière dont différents mécanismes classiques s'intègrent dans le modèle. Le chapitre 5 présente les exemples de mise en œuvre de l'adaptation que nous avons réalisés et qui ont servi de base à cette réflexion.

CHAPITRE 4

Application du patron de conception *Stratégie Autoadaptative* à l'analyse de systèmes

Nous avons vu au chapitre précédent un patron de conception présentant les différents éléments figurant dans un système mettant en œuvre un processus d'adaptation. Un patron de conception ne propose cependant pas de mécanisme ou de modélisation complète. Pour mettre en œuvre les principes qui y sont énoncés, il faut donc avoir recours à des outils, plus ou moins adaptés à la tâche. De plus, on peut trouver également des exemples d'application du patron de conception n'utilisant pas d'outils dédiés, qualifiés alors de mise en œuvre *ad hoc*.

Nous allons utiliser le patron de conception *Stratégie Autoadaptative* pour analyser ces deux types de systèmes. Premièrement, nous nous penchons sur quelques applications autoadaptatives existantes afin de voir de quelle manière elles mettent en œuvre le modèle décrit dans le patron. Deuxièmement, nous étudions certains des mécanismes utilisés pour construire des systèmes adaptatifs et voyons de quelle manière ils s'inscrivent dans le patron de conception. Nous pratiquons cette analyse à la lumière des cinq grands aspects identifiés dans le patron de conception *Stratégie Autoadaptative* :

- *Variété algorithmique.* Elle est représentée par le composant Strategy.
- *Observation.* Elle est représentée par le composant InformationGateway, prenant à la fois en charge l'observation de l'environnement et l'introspection.
- *Contrôle.* Il est représenté par le composant Controller.
- *Gestion de la transition.* Elle comprend notamment la transmission d'informations d'état lors de la transition, représentée par le composant StateAdapter, ainsi que le mécanisme utilisé pour assurer le changement de stratégie.
- *Assemblage.* L'assemblage des sous-systèmes est effectué par le composant

AdaptiveStrategy.

Dowling *et al.* [DSC+99] ont comparé différentes implémentations d'un même système adaptatif, utilisant trois techniques différentes : une conception à partir de patron de conception, l'utilisation de bibliothèques dynamiques, et l'utilisation d'un langage réflexif. Boinot [BMN+00] propose une approche déclarative à l'adaptation, en mettant en exergue sa clarté et sa facilité d'évolution par rapport à une version *ad hoc* basée sur des patrons de conception.

Une confusion dans la notion de patron de conception semble apparaître ici. Les patrons de conception, dans les deux exemples cités ci-dessus, sont mis en concurrence avec des techniques logicielles comme un langage réflexif ou encore une approche déclarative de l'adaptation. Or, un patron de conception dépasse ce niveau pour réfléchir sur des problèmes de plus haut niveau. Que les modèles qu'on en tire soient ensuite implémentés de manière *ad hoc* par un langage classique, ou bien par un langage plus spécialisé ne change rien au patron.

Le patron de conception *Stratégie Autoadaptative* propose un cadre permettant de concevoir et d'analyser des systèmes autoadaptatifs, indépendamment de la manière dont ils sont mis en œuvre.

4.1 Analyse d'applications

On retrouve le patron de conception *Stratégie Autoadaptative* dans des systèmes adaptatifs existants, qui n'utilisent pas forcément de mécanismes ou de plate-formes spécialisés.

4.1.1 Stockage

Les besoins en stockage augmentent continuellement, et le coût de maintenance d'un système de stockage représente maintenant plusieurs fois son coût d'achat. Une des pistes de recherche envisagées consiste en l'administration automatique des systèmes de stockage [BGM+96]. Cette administration automatique s'effectue entre autres par le biais de mécanismes autoadaptatifs intégrés au système.

Par exemple, le projet ISTORE [BOK+99] fournit un système de stockage autoadaptatif modulaire avec une approche couplant un support matériel et une plate-forme logicielle. Les composants matériels de stockage se présentent sous la forme de briques de base, proposant des fonctionnalités actives d'introspection et de surveillance de l'environnement. Une plate-forme logicielle collecte les données de ces briques de base et les présente aux applications sous une forme inspirée des bases de données relationnelles. Ainsi, il est possible de définir des *vues* spécifiques, rassemblant les informations de différents capteurs, ainsi que des déclencheurs (*trigger*) invoquant automatiquement des actions en réaction à des événements particuliers. Pour un certain nombre de tâches communes aux applications manipulant de grands

volumes de données, la plate-forme ISTORE permet la génération automatique du code d'adaptation, tel que la réplication automatique de données, à partir de la spécification sous la forme de contraintes d'intégrité sur la pseudo-base de données contenant les observations des objectifs du système.

L'adaptation dans ISTORE a pour objectif l'administration automatique des systèmes de stockage de données. Les différents aspects du patron de conception *Stratégie Autoadaptative* sont couverts par ce système :

- *Variété algorithmique.* Elle est présente dans la structuration en briques de stockage essentielles, présentant des propriétés différentes (disques, mémoire, bandes, etc).
- *Observation.* Le mécanisme mis en œuvre permet d'obtenir les deux aspects du composant InformationGateway, à savoir l'observation active par le biais des déclencheurs et la mise à disposition (composante passive) des informations collectées.
- *Contrôle.* Le mécanisme de contrôle est à développer spécifiquement par les utilisateurs de la plate-forme pour correspondre spécifiquement à leurs besoins. Cependant, le mécanisme de génération automatique de code d'adaptation remplit ce rôle pour les besoins les plus courants.
- *Gestion de la transition.* Le mécanisme de transition profite également de l'inspiration forte provenant des bases de données, et est centré autour de la notion de transaction. L'aspect de transmission des informations n'est pas explicité, mais est étroitement lié aux fonctions premières du système (stockage d'informations).
- *Assemblage.* Le système fournit une abstraction de stockage aux utilisateurs, isolant les mécanismes sous-jacents, et notamment ceux de l'adaptation effectuée.

4.1.2 Systèmes d'exploitation

La complexité du réglage des systèmes d'exploitation en fait également un type d'application pouvant bénéficier de l'autoadaptation. Ainsi, le système Synthesis [MP90] est un des premiers systèmes d'exploitation intégrant cette notion. Il introduit une politique adaptative d'ordonnancement des processus, s'inspirant du modèle électronique de la boucle à verrouillage de phase. Plus proche de notre approche, le système VINO [SS98, SESS94] fournit un framework générique, basé sur la notion de *graft* (signifiant *greffon*, petit module pouvant être inséré dans le système), permettant l'autoadaptation de différents mécanismes tels que la gestion de la mémoire, la gestion des disques ou la gestion des interruptions.

L'objectif de VINO est l'amélioration des performances du système, caractérisée par les latences des applications. On retrouve en partie les différents aspects du patron de conception *Stratégie Autoadaptative*.

- *Variété algorithmique.* Les *greffons* représentent le mécanisme d'encapsulation

des différentes stratégies disponibles.

- *Observation.* L'aspect d'observation est implanté au niveau de chacun des modules du système, qui maintiennent un historique des événements importants. Un *greffon* est chargé ensuite de centraliser les différentes informations et de les mettre à disposition dans une base de données. Une analyse hors-ligne est pratiquée afin de déterminer plus finement une caractérisation du mode normal de fonctionnement, ce qui permet de détecter plus efficacement les anomalies.
- *Contrôle.* L'aspect contrôle est présent au niveau du *greffon* d'observation, qui peut déclencher une reconfiguration de certains modules. De plus, des simulations peuvent être menées sur la base de différents scénarios afin de déterminer la meilleure configuration vers laquelle s'orienter. On observe donc un contrôle à deux niveaux : un contrôle immédiat, visant à répondre à des cas prévus par les concepteurs du système, et un contrôle continu visant à déterminer des évolutions à plus long terme.
- *Gestion de la transition.* La manière dont ce système gère les transitions n'est pas précisée dans les documents dont nous disposons.
- *Assemblage.* L'ensemble des sous-systèmes est intégré dans le noyau du système. Cependant, on n'observe pas de liaison plus spécifique entre eux.

4.1.3 Protocoles

Dans le domaine des réseaux, les protocoles adaptatifs [HMS99] soulignent également l'importance d'une distinction nette entre adaptation et exécution. Dans le modèle proposé par Hadzic *et al.* [HMS99], des fonctions peuvent être dynamiquement insérées ou supprimées au niveau des chaînes de traitement en réponse aux modifications de l'environnement d'exécution.

La figure 4.1 présente la structure générale d'un module de protocole adaptatif. On y voit apparaître l'aspect d'observation sous la forme d'un sous-module Moniteur. Ce sous-module communique avec celui de décision, qui représente l'aspect contrôle du système. L'aspect transition est géré au niveau du module de signalisation, qui s'assure que la modification de protocole est prise en compte par les systèmes distants concernés avant d'activer la nouvelle configuration.

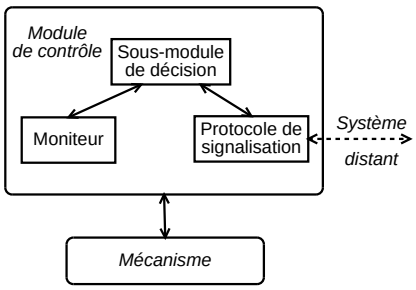

FIG. *4.1* — Structure d'un module de protocole adaptatif

Les protocoles adaptatifs visent à accroître l'adéquation des mécanismes utilisés, notamment pour augmenter les débits possibles. On y retrouve les différents aspects soulignés par le patron de conception *Stratégie Autoadaptative*.

- *Variété algorithmique.* La variété algorithmique est induite par les multiples fonctions pouvant être insérées au niveau de la chaîne de traitement.
- *Observation.* L'observation est isolée dans un module dédié, Moniteur.
- *Contrôle.* Le contrôle est effectué par un module de décision qui utilise, dans un des exemples proposés, des actions associées au dépassement de valeurs de compteurs d'événements.
- *Gestion de la transition.* Un aspect particulier de la transition pour les protocoles est le besoin de synchroniser les mécanismes utilisés par les deux extrémités de la transmission. Cet aspect est réalisé par l'utilisation d'un protocole de signalisation spécifique.
- *Assemblage.* Le point d'entrée du système ne change pas, l'adaptation consistant en l'ajout ou la modification d'éléments au sein de la chaîne de traitement.

4.2 Analyse de mécanismes

Nous allons maintenant étudier quelques uns des outils et techniques pouvant être utilisés comme support au développement d'applications autoadaptatives, et les aspects du patron de conception *Stratégie Autoadaptative* qu'ils couvrent.

4.2.1 Niveau analyse/modélisation

Lors de l'analyse d'un système autoadaptatif, on peut utiliser des patrons de conception pour fournir un cadre de réflexion. Au moment du raffinement du modèle développé, on va avoir recours à des méthodes plus formelles de modélisation, telles que les graphes de fonctionnalité [vGBS01].

Braux et Noyé [BN99] proposent, par une composition de patrons de conception, de fournir un cadre de conception de systèmes capables de changer dynamiquement de comportement, rejoignant ainsi notre modèle. C'est un exemple de présence du patron *Stratégie Autoadaptative* à l'intérieur d'un autre patron. Un objet est défini par un triplet (identité, état, comportement), l'état de l'objet consistant en l'ensemble des valeurs de ses variables d'instance. Les comportements de l'objet peuvent être modifiés en réaction à un changement de son état ou de l'état d'un autre objet, par un mécanisme relevant du patron de conception *Observer*. L'avantage de cette approche est de donner un cadre précis d'analyse, au prix d'une spécialisation du modèle : l'adaptation est uniquement *on change*, et les critères de changements sont contraints d'être traduits dans des attributs d'un objet. De plus, l'aspect de contrôle n'est pas clairement isolé. Les aspects couverts par ce modèle sont donc les suivants :

- *Variété algorithmique.* Elle est exprimée, comme dans le patron de conception *Stratégie Autoadaptative*, par le biais du patron *Strategy*.

- *Observation.* L'aspect d'observation est traité, comme dans le patron de conception *Stratégie Autoadaptative*, par le biais du patron *Observer* qui notifie la stratégie d'un changement d'état.
- *Contrôle.* L'aspect de contrôle est placé au sein de la stratégie définie par le patron *Strategy*, et non séparé.
- *Transition.* Les enjeux relatifs à la transition ne sont pas traités.
- *Assemblage.* La stratégie, qui réunit les aspects de contrôle et d'application de l'adaptation, constitue le point d'entrée du système adaptatif.

De la manière analogue, le patron de conception *State* est une extension du patron de conception *Strategy* qui permet de lier un comportement à chacun des états du système. On peut donc utiliser ce type de patron pour effectuer une adaptation *on change* à l'intérieur d'un système. Cette approche constitue une spécialisation du patron de conception *Stratégie Autoadaptative*, mais n'effectue pas de séparation claire entre observation et contrôle de l'adaptation.

Les graphes de fonctionnalités étendus [vGBS01], présentés dans 2.6.5, présentent un autre moyen de modéliser l'adaptation. Le concept clé de cette approche est celui de point de variation, qui identifie dans le système l'emplacement des fonctionnalités qui seront soumises à l'adaptation. La figure 2.6 page 68 présente ainsi un modèle de client de courrier électronique où sont identifiés plusieurs points de variation (pour le protocole d'accès au courrier, la fonctionnalité d'édition, etc). Cette approche permet donc d'établir un cadre à l'application du patron de conception *Stratégie Autoadaptative*, en précisant à la fois l'emplacement de la fonction et les différentes variantes possibles.

Une fois qu'ils ont été modélisés, il est possible de mettre en œuvre les mécanismes autoadaptatifs par le biais de différents supports techniques, de plus ou moins haut niveau. Nous distinguons quatre niveaux pour ces supports : le niveau intergiciel (*middleware*), le niveau cadre de conception (*framework*), le niveau langage et le niveau système d'exploitation.

4.2.2 Niveau *middleware*

Les intergiciels (*middleware*) constituent une couche logicielle s'intercalant entre les applications et les systèmes d'exploitation, destinée à fournir la concrétisation d'un modèle de distribution. Le domaine des applications distribuées est particulièrement propice au développement d'applications adaptatives [DB00], notamment à cause de la grande hétérogénéité des ressources. La couche d'abstraction fournie par les *middleware* est un support judicieux aux mécanismes communs aux systèmes adaptatifs tels que l'introspection ou les mécanismes de notification. Le principe de réflexivité [Mae87] y est particulièrement important. La réflexion est une manière pour un système d'accéder à une représentation de son état interne, ainsi que de pouvoir le manipuler. On peut distinguer deux niveaux dans un système réflexif [DB00] : le niveau de base, qui remplit les objectifs premiers du système, et le niveau

méta, qui manipule une représentation du système lui-même et peut en modifier le fonctionnement.

OpenORB [DB00, ABE00] est un *middleware* réflexif, qui associe à tout objet un *méta-espace* structuré suivant quatre méta-modèles orthogonaux, qui correspondent à différents aspects du patron de conception *Stratégie Autoadaptative* :

- *Variété algorithmique.* Le modèle d'encapsulation offre les fonctions d'inspection et de changement des méthodes des objets, entraînant ainsi la possibilité de manipuler les variantes des stratégies.
- *Observation.* Le modèle de ressource définit l'accès aux ressources ainsi que leur gestion. Les ressources sont représentées par une hiérarchie d'objets, qui permettent d'obtenir dans les niveaux les plus élevées une vision synthétique des conditions d'exécution.
- *Contrôle.* Le modèle de comportement précise la manière dont sont effectuées les opérations.
- *Transition.* Les enjeux relatifs à la transition ne sont pas traités.
- *Assemblage.* Le modèle de composition définit la manière dont les composants sont interconnectés.

4.2.3 Niveau cadre de conception

Les cadres de conception (*framework*) sont des ensembles d'objets agencés de manière à fournir un cadre logiciel réutilisable pour des types d'applications précis. Il suffit aux programmeurs d'en adapter le fonctionnement au niveau d'emplacements laissés ouverts dans la structure pour obtenir des applications complètes. Cette approche est adaptée à la conception d'applications autoadaptatives, par sa capacité à fournir à la fois un modèle de programmation et un support d'exécution.

Jaws

Jaws [HS99, Hu98] est une plate-forme de développement de serveurs web adaptatifs, construite au dessus de la bibliothèque de communication ACE. Elle vise à fournir les composants essentiels à la mise en œuvre de serveurs web dynamiquement adaptables, capables de réagir aux changements de conditions d'exécution. Pour cela, différentes stratégies essentielles ont été isolées, qui font chacun l'objet d'un traitement particulier dans un sous-système : la gestion des événements, la gestion de la concurrence, la gestion des entrées/sorties, la gestion des protocoles, la gestion d'un système de fichiers virtuel, etc. Les mécanismes proposés sont essentiellement dédiés au mécanisme d'adaptation, traitant les aspect de réification des stratégies ainsi que celui de façade en redirigeant dynamiquement les messages vers les stratégies adéquates.

- *Variété algorithmique.* Différents types de stratégies sont isolés, et leurs déclinaisons sont mises en œuvre par le biais du patron de conception *Strategy*.

- *Observation.* Les mécanismes d'observation ne sont pas directement intégrés à la plate-forme.
- *Contrôle.* La partie contrôle n'est pas spécifiée et doit être élaborée par l'utilisateur de la plate-forme.
- *Gestion de la transition.* L'aspect de transition est spécifié par un patron de conception *Service Configurator* [JS97], qui gère le mécanisme de transition (voir paragraphe 3.5.3). La transmission d'informations d'état entre différentes stratégies n'est pas mentionnée.
- *Assemblage.* Les différents sous-systèmes identifiés constituent autant de systèmes autoadaptatifs, dont le point d'accès est défini par la plate-forme.

MolèNE

MolèNE [SA00] est un *framework* permettant de construire des applications adaptatives dans le domaine des environnements mobiles. L'objectif est de fournir une couche intermédiaire entre les applications et le système d'exploitation permettant de transformer le plus facilement possible des applications classiques en applications prenant en compte le caractère dynamique des environnements mobiles. Dans ce but, MolèNE fournit un ensemble de services et d'outils permettant de mettre en œuvre des applications adaptatives. Les services fournis comportent un service de distribution, qui se charge de déterminer la meilleure localisation possible pour l'exécution des applications et le stockage des données, un service de cache des données. Les outils et services sont implémentés sous la forme de composants. L'implémentation de classes abstraites du *framework* permet aux développeurs de construire des applications adaptatives applicables au domaine des mobiles. L'adaptation dynamique est obtenue en construisant un service à base de composants adaptatifs encapsulant des décisions d'adaptation dynamiques de leur comportement.

- *Variété algorithmique.* Les différentes variantes de mise en œuvre d'un service répondent à une interface commune, définie par un composant Implantation.
- *Observation.* L'aspect d'observation est fourni par le système de détection et de notification, décrit dans le paragraphe 3.5.1, dans lequel on retrouve notamment l'approche active/passive de l'observation.
- *Contrôle.* La stratégie d'adaptation pour un composant est gérée par un automate dont les transitions représentent les actions d'adaptation.
- *Gestion de la transition.* L'aspect de transfert d'état est pris en charge explicitement par un service appelé state adapter, qui peut fonctionner de manière autonome si les informations ne nécessitent pas d'adaptation trop importante, ou bien être spécialisé dans le cas contraire.
- *Assemblage.* On obtient après la mise en œuvre des composants réactifs constituant le point d'entrée au service défini.

4.2.4 Niveau langage

On s'intéresse ici principalement aux langages de programmation orientés objet. Ces derniers sont effectivement les plus répandus actuellement et proposent des fonctionnalités telles que la liaison dynamique et l'encapsulation de données qui permettent d'exprimer plus facilement des mécanismes utilisés dans les systèmes adaptatifs.

Les langages réflexifs

Un langage est réflexif quand une partie de ses caractéristiques (sémantique, architecture de base) est réifiée, dans un objectif de contrôle et de manipulation par un niveau supérieur du programme, que l'on qualifie de niveau *méta*. Un des objectifs majeurs de la mise en œuvre de mécanismes réflexifs est l'autoadaptation des systèmes. À l'intérieur du patron de conception *Stratégie Autoadaptative*, ils s'inscrivent dans plusieurs aspects. L'aspect observation représenté par le composant InformationGateway est le plus évident : le contexte que nous avons défini dans le paragraphe 2.1 comprend notamment le système lui-même, entraînant un besoin d'introspection. La séparation des aspects entre le processus d'adaptation et le processus d'exécution est perceptible dans les langages réflexifs dans la distinction entre le niveau de base et le niveau dit *méta*.

Les langages orientés objet sont – par leurs propriétés d'encapsulation des données, d'héritage, etc. – un support privilégié du concept de réflexion, même si les premiers développements ont été effectués autour de langages fonctionnels tels que Lisp [dRS84]. On peut distinguer deux types d'implémentation de la réflexion au niveau de la programmation objet : les langages basés sur la notion de méta-classe, et ceux basés sur la notion de méta-objet. Dans le premier type [GR83], le comportement d'un objet lors de la réception d'un message est défini dans sa classe et le comportement de la classe est défini dans une méta-classe. Les langages basés sur les méta-objets n'utilisent pas la notion de méta-classe, mais proposent d'étendre la sémantique des objets par l'adjonction d'un méta-objet qui est responsable par délégation du traitement des messages envoyés à l'objet initial. Il peut alors traiter le message lui-même, ou bien le transmettre après d'éventuelles transformations à l'objet original. Les capacités des langages réflexifs en termes de manipulation de leur modèle sont précisées par le *Meta-Object Protocol* (MOP), qui définit les primitives de manipulation du langage.

Par exemple, Iguana [DSC+99] est un modèle de programmation réflexive, appliqué aux langages *java* et *C++*. C'est un modèle méta-objet, permettant d'associer un ou plusieurs méta-objets à un système. Un des objectifs d'Iguana est de permettre d'ajouter facilement des mécanismes réflexifs à un système déjà existant. Le fonctionnement réflexif du système est spécifié par le biais d'un *protocole d'extension* qui associe un méta-objet aux classes adéquates du système considéré.

– *Variété algorithmique*. La variété algorithmique découle de la possibilité de

modifier dynamiquement certaines méthodes d'un objet. La manière dont les différentes variantes sont offertes n'est pas détaillé précisément, mais la réification de méthodes est partie intégrante du langage, du fait de son caractère réflexif.
- *Observation.* L'aspect d'observation découle de la mise en œuvre de l'introspection dans le système. Cependant, une partie de cet aspect reste à la charge du client.
- *Contrôle.* L'aspect contrôle est mis en œuvre dans le méta-objet.
- *Gestion de la transition.* L'adaptation est prise en compte au niveau *méta* mais la manière dont elle est effectuée, notamment au moment de la transition, n'est pas spécifiée.
- *Assemblage.* La façade est réalisée par le *protocole d'extension* qui spécifie la liaison entre le niveau contrôle et le processus d'exécution.

Approche déclarative

Boinot [BMN+00] propose une approche déclarative pour modéliser les différents aspects de systèmes adaptatifs. Cette méthode se rapproche de la programmation par aspects, séparant les aspects contrôle et exécution. Un langage spécialisé est utilisé pour spécifier les comportements à utiliser en fonction des conditions de fonctionnement. Par exemple, l'adaptation d'un encodeur audio disposant de deux types d'encodage (GSM et ADPCM) s'effectue par les commandes suivantes, sachant que les stratégies concrètes GSM et ADPCM héritent de la stratégie abstraite SoundEncoder :

```
adaptclass ExtendedSoundEncoder extends SoundEncoder {
  when (rtpControl.bandwidth < 16) Gsm ();
  when (16 <= rtpControl.bandwidth && rtpControl.bandwidth < 48) Adpcm(2);
}
```

Les conditions d'adaptation sont extraites par le compilateur spécialisé, qui les traduit en un système dédié d'observation et de notification qui est inséré à l'intérieur du système.

- *Variété algorithmique.* Les différentes variantes sont préexistantes dans le système sur lequel on veut appliquer l'adaptation.
- *Observation.* Les mécanismes d'observation sont élaborés par instrumentation à partir des conditions extraites par le compilateur spécialisé.
- *Contrôle.* L'aspect contrôle est isolé dans la déclaration. Après compilation, il se retrouve dans la nouvelle classe créée, ExtendedSoundEncoder au sein de laquelle les mécanismes de réception de notifications et de prise de décision sont intégrés.
- *Gestion de la transition.* La transmission d'informations d'état est l'objet d'un mécanisme de copie simple de certaines informations, ne prenant pas en compte d'éventuels besoins de transformation de l'information.

– *Assemblage.* Le système résultant est matérialisé par la création d'une nouvelle classe intégrant les différentes fonctionnalités.

4.2.5 Niveau système d'exploitation

Il faut faire une distinction entre les systèmes d'exploitation proposant des mécanismes d'adaptation dynamique, et les systèmes d'exploitation mettant en œuvre des mécanismes d'adaptation dynamique [Cah96]. Cependant, on peut souligner que les mécanismes qu'utilise un système d'exploitation adaptatif, notamment tous les outils de *monitoring*, sont susceptibles d'être utilisés par des applications désirant intégrer un comportement adaptatif. Nous avons vu au paragraphe 4.1.2 des exemples de systèmes d'exploitation adaptatifs. Nous présentons maintenant un système d'exploitation proposant des fonctionnalités destinées à servir de support à des applications autoadaptatives.

Odyssey [NSN+97] est un système d'exploitation visant à fournir un support pour la conception d'applications adaptatives. Divers mécanismes sont proposés pour arriver à cette fin. Tout d'abord, un système d'observation est mis en place, qui permet aux applications d'exprimer leurs besoins en termes de ressources et qui renvoie des notifications appropriées. Le mode d'adaptation est orienté vers la notion de transfert de données. Odyssey définit ainsi des modules de gestion (*warden*) spécifiques à différents types de donnéee (vidéo, image, etc.), qui sont chargés de réaliser l'adaptation du mode de transfert.

– *Variété algorithmique.* Les variantes algorithmiques ne sont pas explicitées, elles peuvent être propres à chaque type de données.
– *Observation.* L'aspect observation est explicitement intégré sous la forme d'un système de notification.
– *Contrôle.* L'aspect contrôle est partagé entre les applications, qui spécifient leurs besoins et les réactions aux événements, et les modules de gestion des données qui mettent en œuvre le mécanisme de changement.
– *Gestion de la transition.* Les problèmes de transition ne sont pas mentionnés.
– *Assemblage.* Les points d'entrées de l'adaptation sont définis par les modules (*warden*).

4.3 Conclusion

Nous avons présenté dans le chapitre 3 le patron de conception *Stratégie Autoadaptative*, qui définit la manière dont un système peut mettre en œuvre les différents aspects (introspection/observation, contrôle, transition) liés au processus d'autoadaptation. Dans ce chapitre, nous avons voulu voir de quelle manière le patron de conception proposé s'applique à des systèmes déjà existants et comment sont mis en œuvre certains aspects dans des cas spécifiques. Une synthèse de cette analyse est présentée dans le tableau 4.1. De plus, nous avons étudié différentes techniques

permettant de mettre en œuvre des systèmes autoadaptatifs et identifié les aspects du patron de conception qu'elles permettent de réaliser. Nous avons synthétisé nos observations dans le tableau 4.2.

Système	**Variété algorithmique**	**Observation**	**Contrôle**	**Transition**	**Assemblage**
ISTORE	Briques de stockage	Active et passive	Génération automatique de code d'adaptation	Non spécifié	Abstraction de stockage
VINO	*Greffons*	Active, et centralisée dans une base de données	Immédiat (réaction) ou à long terme (simulations)	Non traitée	Non explicite
Protocoles adaptatifs	Variantes de la chaîne de traitement	Active et passive via un module dédié	Isolé dans un module de décision	Gestion d'une synchronisation entre extrémités d'une connexion	Point d'entrée inchangé

TAB. *4.1* — Analyse de systèmes adaptatifs

Mécanisme	**Variété algorithmique**	**Observation**	**Contrôle**	**Transition**	**Assemblage**
Composition de patrons de conception	Patron *Strategy*	Patron *Observer*	Non isolé	Non traité	Via le patron *Strategy*
OpenORB	Modèle d'encapsulation	Modèle de ressource	Modèle de comportement	Non traité	Modèle de composition
Jaws	Patron *Strategy*	Non spécifié	Non spécifié	Patron *Service Configurator* pour le mécanisme	Identification en sous-systèmes
MolèNE	Dérivation d'un composant Implementation	Système de Détection et de Notification (SDN)	Automate à états	Prise en charge explicite par un service state adapter	Composant
Iguana	Réification de méthodes	Introspection	Niveau *méta*	Non spécifiée	*Protocole d'extension*
Approche déclarative	Préexistante	Instrumentation	Déclaratif	Copie de variables	Création d'une nouvelle classe
Odyssey	Non explicité	Système de notification	Défini par les applications	Non spécifié	Modules (*warden*)

TAB. *4.2* — Analyse de mécanismes d'adaptation

CHAPITRE 5

Réalisations

Nous avons mis en application le modèle proposé au chapitre 3 dans deux domaines principaux : le domaine des tris, et celui des caches web. Ces deux domaines présentent en effet des propriétés intéressantes au regard de l'adaptativité, notamment la multiplicité des solutions et la variabilité des conditions d'exécution.

Après avoir présenté le principe du tri adaptatif, nous décrivons l'application du modèle décrit par le patron de conception *Stratégie Autoadaptative*. Nous fournissons ensuite les résultats des mesures que nous avons réalisées, visant à déterminer l'opportunité de la mise en place d'un mécanisme d'adaptation et nous voyons les apports de cette expérience.

Nous avons effectué nos travaux dans le domaine des caches web sur la base de deux outils : d'une part un simulateur appelé *Saperlipopette !*, pour lequel nous avons testé l'impact d'un changement de stratégie de remplacement. D'autre part, la plate-forme Jigsaw dans laquelle nous avons implanté un mécanisme de stockage adaptatif. Le paragraphe 5.2 traite de la manière dont nous avons pu mettre en œuvre l'adaptation dans le simulateur *Saperlipopette !* en utilisant le patron de conception *Stratégie Autoadaptative*, et en particulier le traitement de la phase de transition entre deux stratégies. Le paragraphe 5.3 décrit les modifications que nous avons apportées au système Jigsaw afin de mettre en œuvre un mécanisme de stockage adaptatif.

5.1 Le tri adaptatif

Nous avons voulu travailler sur un exemple simple afin de mettre en évidence certains principes de l'adaptation. Nous nous sommes donc penchés sur le cas des algorithmes de tri de nombres entiers. Ce domaine présente en effet une grande variété algorithmique, et a déjà fait l'objet de nombreuses recherches.

5.1.1 Principe

Le tri est une des opérations informatiques les plus courantes. En conséquence, de nombreux algorithmes ont été mis au point pour répondre à ce problème. La pertinence d'un algorithme dépend de plusieurs critères comme le contexte de l'application, le critère de tri, les structures de données utilisées, etc. Aksit et Tekirnedogan [AT98, AT99] ont proposé une modélisation du domaine du tri que nous avons déjà présentée au paragraphe 2.6.2.

Nous avons restreint notre étude aux tris sur des entiers stockés dans une structure de données simple de type liste. Le nombre de mise en œuvres différentes reste suffisamment élevé pour envisager la mise en place d'un mécanisme d'adaptation. L'objectif de l'adaptation est de minimiser le temps d'exécution d'un tri. Pour cela, on ne peut se baser simplement sur des propriétés intrinsèques des algorithmes telles que leur complexité ou le nombre moyen de comparaisons effectuées [Knu73]. Comme le soulignent les auteurs du programme adaptatif de transformée de Fourier FFTW [FJ98], de nombreux facteurs spécifiques à chaque ordinateur peuvent introduire un biais entre la mesure classique de complexité et la vitesse d'exécution, notamment la présence d'un cache anté-mémoire au niveau du processeur. Nous avons donc décidé de commencer par effectuer des mesures sur différents ensembles, mais sur une architecture unique.

Nous avons mis en œuvre différents algorithmes en utilisant le langage java. Nous avons procédé à une réification des algorithmes, utilisant le patron de conception *Strategy* tel que décrit dans [GHJV95]. Une fois ces algorithmes disponibles, nous avons construit le mécanisme destiné à assurer l'adaptation du tri en fonction des caractéristiques des données.

Parmi les différents algorithmes implémentés figurent le tri rapide (Quicksort) [Hoa61], le tri à bulles, le tri par insertion, le tri par sélection, le tri shell, le tri par fusion, le tri par tas, le tri par dénombrement, le tri par base et le tri par paquet [Knu73].

5.1.2 Mise en œuvre de l'adaptation

La mise en œuvre de l'adaptation est effectuée en appliquant le patron de conception *Stratégie Autoadaptative* à la stratégie abstraite de tri. Les différentes stratégies concrètes étant mises en œuvre par le biais d'un patron de conception *Strategy*, nous pouvons faire évoluer ce modèle directement.

Nous avons donc créé un composant AdaptiveSort, point d'entrée aux fonctionnalités de tri, ainsi qu'un contrôleur simple Controller obtenant ses informations d'un composant SortContext qui joue le rôle de l'InformationGateway. La figure 5.1 présente la structure du système obtenu.

On peut en particulier identifier deux variantes pour le contrôleur : présenter, via l'InformationGateway représenté par le composant SortContext, des points d'entrée

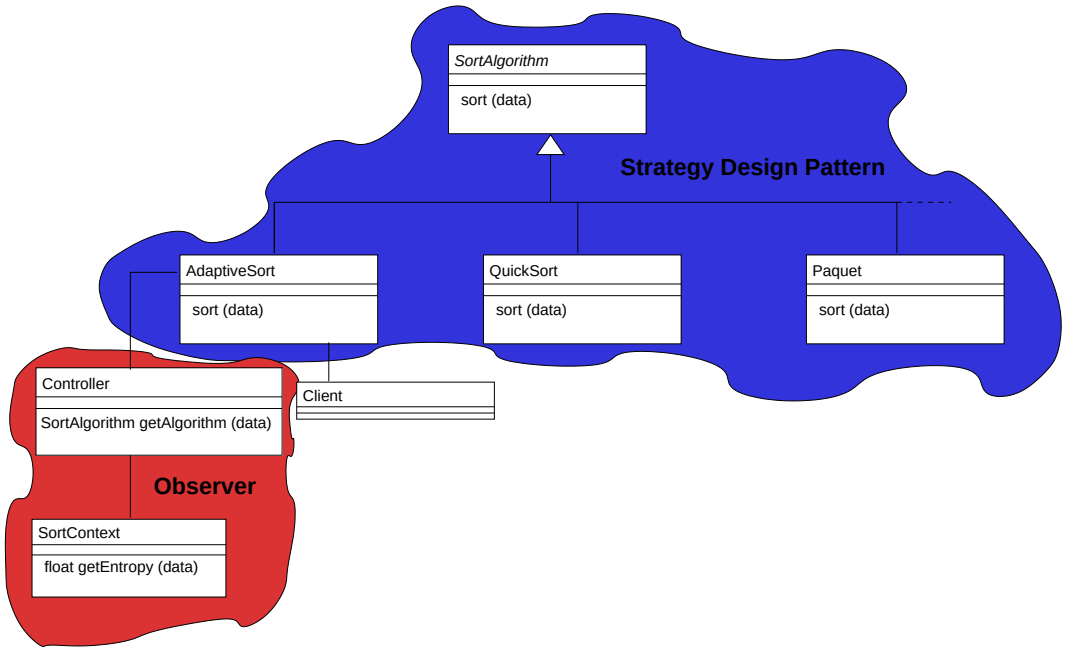

FIG. *5.1* — Structure du tri adaptatif

permettant à l'utilisateur de préciser lui-même certaines caractéristiques de l'ensemble à trier (valeurs maximales, étendue des données, etc.), ou bien implémenter un algorithme de détermination des caractéristiques de l'ensemble à trier, ce qui assurerait un fonctionnement totalement autonome du contrôleur.

On voit apparaître ici une potentialité d'évolution par raffinement du modèle. On peut ainsi dans un premier temps utiliser la première option, et si l'opportunité (en termes de besoins ainsi que de capacité d'implémentation) se présente, compléter le système par un module de détermination des caractéristiques. Dans la figure 5.1, nous avons introduit une fonction `getEntropy` qui retourne la valeur d'entropie de Shannon [SW49] de l'ensemble à trier. C'est en effet un des critères possibles permettant de déterminer des caractéristiques de l'ensemble de données.

Cette démarche rentre bien dans le cadre de la loi d'évolution du logiciel. La première étape conserve une interaction avec un opérateur, ou bien avec le concepteur du programme, afin d'aider à la prise de décision. La seconde étape va vers une autonomie plus grande du système de tri.

5.1.3 Mesures

Nous avons procédé à la mesure des durées d'exécution des différents algorithmes sur une seule architecture, par instrumentation du code. Différents ensembles de test ont été utilisés, générés spécifiquement de manière à présenter une caractéristique particulière. Nous avons ainsi étudié les performances des algorithmes de tri sur des

ensembles ne présentant pas de valeurs en doublon, sur des ensembles présentant des doublons, sur des ensembles comportant au plus quatre valeurs distinctes, sur des ensembles déjà triés et sur des ensembles triés dans l'ordre inverse (tableau 5.1).

Les cas les plus pathologiques comme celui de l'ensemble déjà trié ou trié dans l'ordre inverse nous ont permis de mesurer l'impact du temps de détermination de la caractéristique permettant d'appliquer l'algorithme optimal (pas de tri, ou bien simple inversion). Ainsi, dans le tableau 5.1, nous présentons les performances comparées de quatre algorithmes différents sur un ensemble trié dans l'ordre inverse. Les quatre algorithmes utilisés sont le tri QuickSort, le tri par tas, le tri shell et un simple algorithme d'inversion. La dernière ligne du tableau présente le temps d'exécution de la fonction de caractérisation, qui détermine si l'ensemble est trié dans l'ordre inverse ou non. Ce temps d'exécution est loin d'être négligeable, et son cumul avec le temps d'exécution de l'algorithme le plus adapté dans ce cas (inversion de l'ensemble) est supérieur au temps d'exécution de l'algorithme QuickSort. Le coût de l'adaptation est donc ici trop élevé, même en prenant en compte uniquement sa composante observation.

	1000	10000	100000	1000000
QuickSort	0	8	97	117
Tas	2	50	700	8319
Shell	2	29	401	5295
Inverse	0	0	5	59
EstInverse	0	0	7	79

TAB. *5.1* — Temps d'exécution (en secondes) de différents algorithmes suivant la taille des ensembles triés dans l'ordre inverse

5.1.4 Conclusion

La modèle utilisé nous semble valide, mais l'exemple choisi est inadéquat pour notre étude. En effet, l'algorithme de tri QuickSort est trop efficace sur des ensembles de tailles moyenne à grande pour justifier la mise en œuvre d'un système adaptatif, mis à part peut-être la possibilité d'utiliser une version parallèle sous certains critères. Des travaux [TS92] nous conduisent à penser qu'une distinction est également possible sur des ensembles de très grande taille, mais nous ne disposions pas des moyens et du temps nécessaire pour mener ces expériences. Cependant, nous avons acquis une expérience sur le processus d'adaptation *on action*. De plus, nous avons pu dégager la composante du coût de l'adaptation dû à la caractérisation des données à trier. Sur ce type d'exemples, il est parfois plus long d'essayer de caractériser les données que d'appliquer une politique générique.

5.2 *Saperlipopette !*

Nous avons utilisé un simulateur de caches web afin de procéder à une évaluation du modèle d'autoadaptativité dans le domaine des caches web. Nous présentons ici la structure de *Saperlipopette !*, puis nous poursuivons par la description des évolutions que nous lui avons apportées.

5.2.1 Présentation

Saperlipopette ! [Pie99] est un simulateur de caches web distribués à événements discrets. Un de ses objectifs est d'évaluer et de comparer diverses configurations de caches web distribués, tant en termes de placement qu'en termes de dimensionnement. Nous avons utilisé cette plate-forme pour tester l'approche autoadaptative appliquée à la stratégie de remplacement.

5.2.2 Évolutions du logiciel

Dans un premier temps, nous avons procédé à une amélioration du modèle de transport, ainsi qu'à la modification de l'architecture des politiques internes du simulateur afin de les rendre plus flexibles [Car98].

Nous avons pour cela intégré au simulateur la prise en compte du partage des ressources réseaux. En effet, dans la version dont nous disposions, la bande passante sur une connexion est une constante, quel que soit le nombre de requêtes ou réponses y transitant. Cette mise en œuvre part d'une hypothèse d'un débit dû au cache faible vis-à-vis du débit total du réseau. Cependant, cette hypothèse ne tient plus si l'on considère une architecture où le cache dispose de liens dédiés. La mise en œuvre de cette modification a alors consisté à intégrer la possibilité de modifier l'échéance d'un événement afin d'appliquer aux messages un retard calculé en fonction de l'occupation des connexions.

Nous avons également modifié l'architecture des politiques internes du simulateur. Le principal apport de cette modification a été une plus grande modularité, permettant notamment de créer des stratégies (de remplacement, de coopération, etc.) plus facilement paramétrables. Jusqu'ici en effet l'intégration d'une nouvelle stratégie imposait de réécrire une partie du code de lecture du fichier de configuration. Les améliorations apportées ont rendu plus aisée l'extension des stratégies. Sur cette base, nous avons ensuite mis en œuvre la modification dynamique de la stratégie de remplacement du simulateur, afin de vérifier l'impact d'un changement sur les performances du système, et également d'acquérir une expertise sur les concepts d'autoadaptation que nous désirons étudier.

Nous avons étudié quatre stratégies de remplacement censées présenter, d'après notre étude du paragraphe 1.7.7, des comportements suffisamment distincts suivant les données qu'elles peuvent avoir à traiter. Nous avons ainsi mis en œuvre les stra-

tégies *Size*, *GreedyDualSize* et *LRV* (voir le paragraphe 1.7.7) qui venaient s'ajouter à la stratégie *LRU* déjà présente dans le simulateur.

Une première étape de simulation a consisté à mesurer les taux de réussite des différentes stratégies en fonction de la taille du cache. Les résultats de ces mesures sont présentés dans la figure 5.2. On observe que la stratégie *GreedyDualSize* offre de meilleures performances. Cependant, lorsque l'on étudie le taux de réussite pondéré par la taille des documents, présenté dans la figure 5.3, on constate que les stratégies *LRU* et *LRV* présentent de meilleurs résultats.

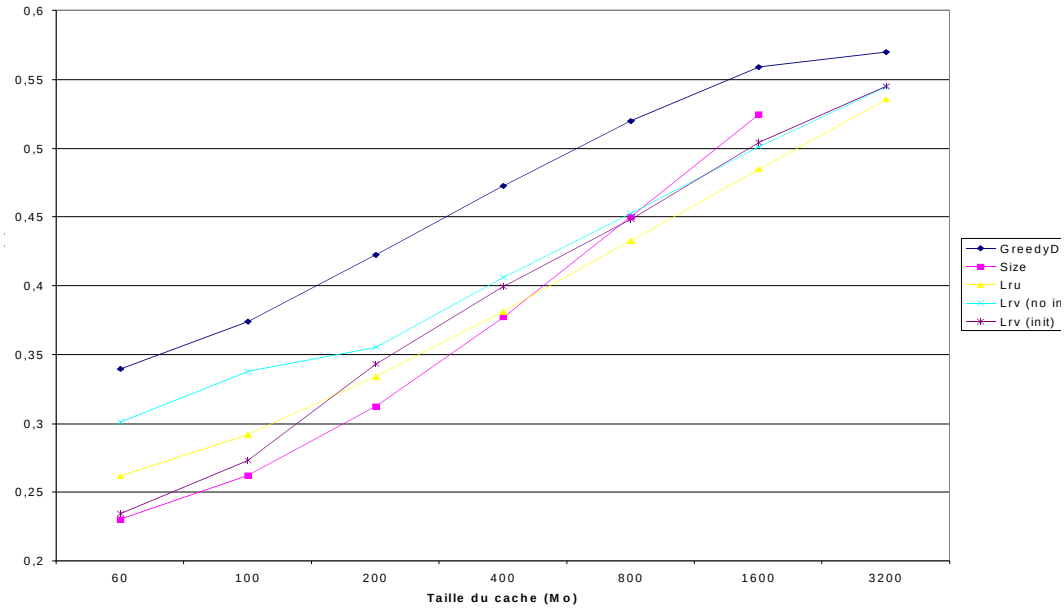

FIG. 5.2 — Taux de réussite en fonction de la taille du cache

Désirant simplement valider le modèle de stratégie autoadaptative, nous ne nous sommes pas intéressés à l'implémentation de l'algorithme de choix, nous contentant alors d'un simple choix aléatoire. La figure 5.4 compare les performances en termes de taux de réussite entre la stratégie aléatoire et trois autres stratégies, pour un même jeu de traces[1] et après un temps de chargement du cache. Nous disposons donc de trois graphes représentant chacun la différence en points de pourcentages entre la performance d'une stratégie fixe et celle de la stratégie aléatoire. Une valeur négative indique donc une augmentation des performances dues à la stratégie aléatoire. Nous avons porté en abscisse, parallèlement à l'axe du temps, les noms des stratégies utilisées successivement par la stratégie aléatoire. Pour clarifier le schéma, nous n'avons pas représenté les différences pour chacune des stratégies que lorsque la stratégie aléatoire a la même valeur, dans l'une des trois phases de fonctionnement décrites plus bas. Par exemple, la différence entre la stratégie aléatoire et la stratégie *LRU* n'est représentée que lorsque la stratégie aléatoire est en phase de transition

[1] Nous avons utilisé le jeu de traces DEC [KMM97].

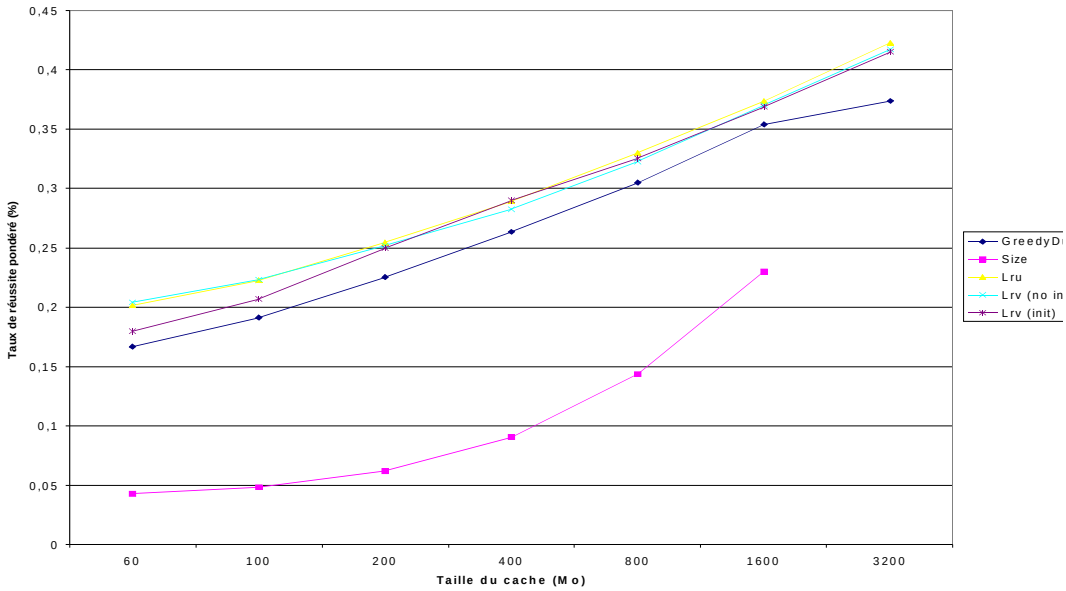

FIG. 5.3 — Taux de réussite pondéré par la taille en fonction de la taille du cache

vers *LRU*, utilise seulement *LRU* ou est en phase de transition depuis *LRU*. Durant les autres périodes, nous avons fixé à zéro la différence.

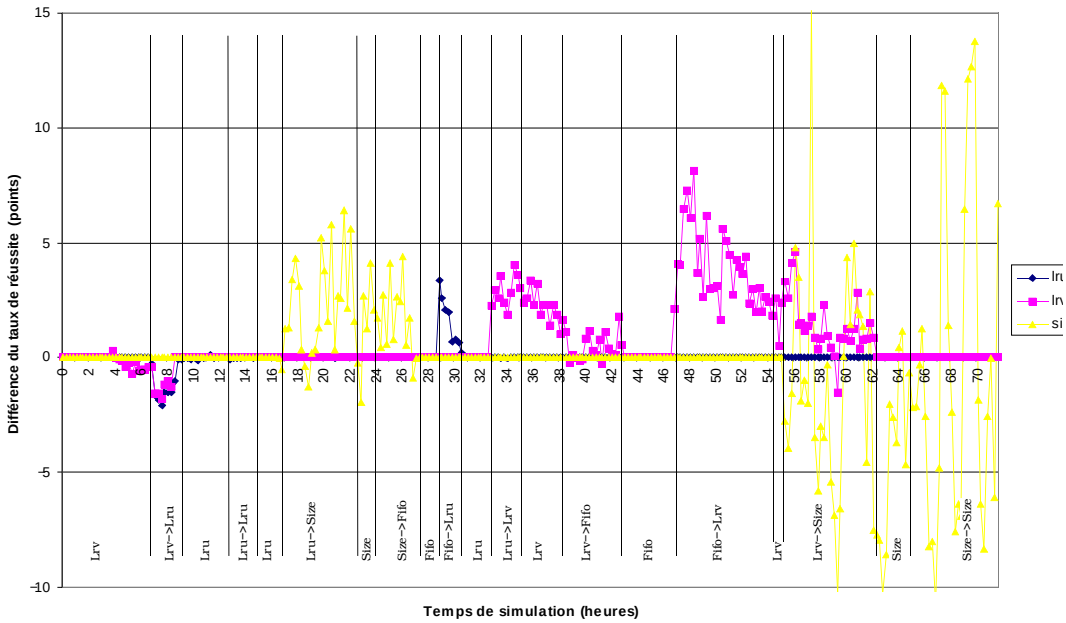

FIG. 5.4 — Taux de réussite avec changement aléatoire de stratégie

Si l'on regarde le taux de réussite pondéré par la taille des documents, nous nous apercevons d'une grande augmentation des performances de la stratégie aléatoire par

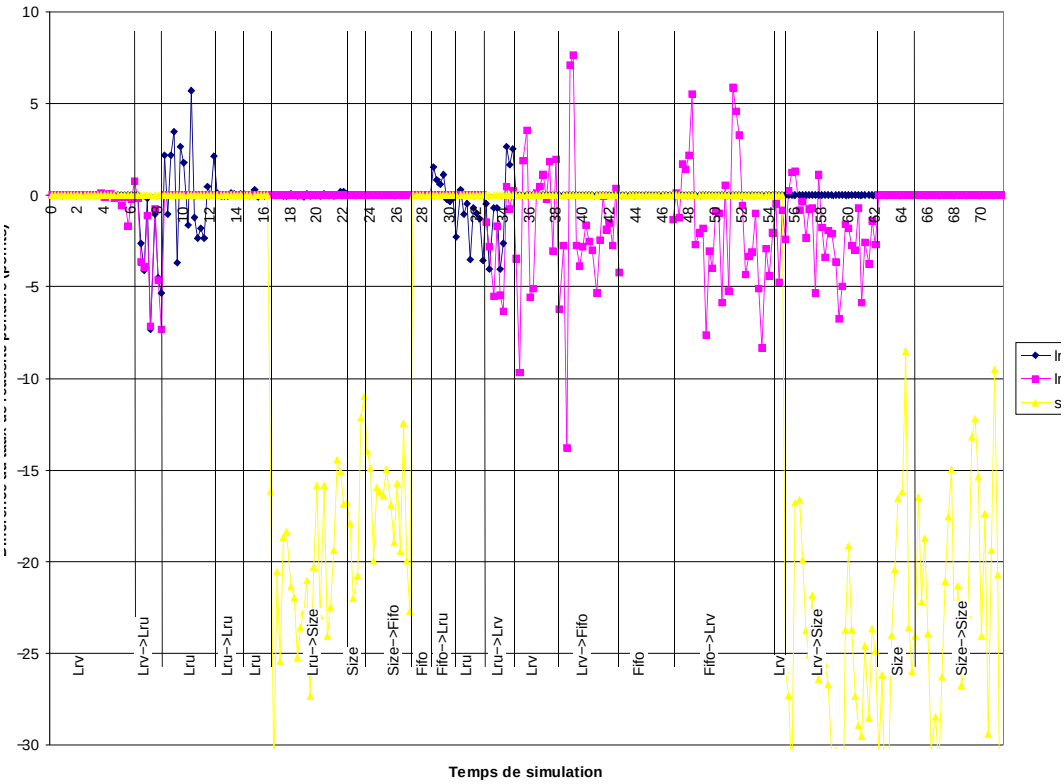

FIG. 5.5 — Taux de réussite pondéré par la taille avec changement aléatoire de stratégie

rapport à la stratégie *Size*. Cette différence est due au fait que la stratégie *Size* peut être amenée à supprimer de gros documents accédés fréquemment, car elle ne prend pas en compte les dates ou fréquences d'accès. Ainsi, lors de la transition depuis *LRU* vers *Size*, au temps 17, le stratégie aléatoire profite de l'état du cache laissé par *LRU*.

5.2.3 Utilisation du patron de conception

Le patron de conception n'a pas été explicitement utilisé, la mise en œuvre étant antérieure à sa définition. Cependant, il est possible d'utiliser le patron pour analyser le code qui a été produit et constater dans quelle mesure on retrouve sa structure dans le système réalisé.

La réification des différents algorithmes est déjà effectuée au sein du simulateur original, et a été améliorée par la première partie du travail présentée au paragraphe 5.2.2. Mettant en œuvre le patron de conception *Strategy*, la stratégie abstraite de remplacement est définie par une classe Replacement fournissant une interface permettant principalement d'avertir des événements intervenant au sein du cache (insertion de document, suppression de document, etc.) et d'effectuer une suppression de document. La modification a consisté en la définition d'une classe Replacement-

Factory, utilisant le patron de conception *Factory*, qui permet de faciliter la gestion des différentes variantes de la stratégie. Différentes stratégies concrètes peuvent alors s'inscrire dans ce cadre : Lru, Lrv, Size.

À partir de cette structure, nous avons créé une classe ReplSwitch qui intègre les rôles des composants AdaptiveStrategy et StateAdapter. C'est en effet le point d'accès principal utilisé par le système de cache, qui se présente comme une stratégie de remplacement à part entière mais joue surtout le rôle d'un redirecteur (ou d'une *Facade* dans le langage des patrons de conception [GHJV95]). De plus, la gestion de la transition d'une stratégie à une autre est également effectuée à ce niveau.

L'aspect Controller est implémenté par la génération au niveau du simulateur d'un événement à une date aléatoire, entraînant l'activation d'un nouveau choix de stratégie. Cet événement se concrétise sous la forme d'une instance de la classe ReplSwitchHdr, dont la convention de nommage est tirée du code existant de *Saperlipopette !*.

La mise en œuvre choisie pour le Controller étant d'effectuer un changement aléatoire, nous n'avons pas mis en œuvre l'aspect d'observation et d'introspection du système.

Gestion de la transition

La transition d'une stratégie à une autre est gérée par la classe ReplSwitch, comme nous l'indiquions dans le paragraphe 3.5.3. Nous utilisons une méthode de fonctionnement parallèle des deux stratégies, illustrée par la figure 5.6.

La stratégie ReplSwitch maintient une référence à une stratégie courante, appelée repl1, et à une stratégie à activer appelée repl2. Tant que l'activation de repl2 n'a pas été validée, tous les appels sont transmis à repl1. Lorsque repl2 a été validée, ReplSwitch va transmettre les informations sélectivement à l'une ou l'autre stratégie. Les demandes de suppression de documents sont transmises à repl1 jusqu'à ce que la liste des documents qu'il gère soit vide. Pendant ce temps, les documents nouvellement insérés dans le cache sont enregistrés auprès de la nouvelle stratégie. En cas d'accès à un document géré par l'ancienne stratégie, sa référence est transmise à la nouvelle stratégie, et supprimée de l'ancienne. Une fois que l'ensemble des documents gérés par l'ancienne stratégie repl1 a été épuisé, cette dernière peut être désactivée. On obtient durant la phase de transition une gestion de type *LRU* où l'on épuise les documents accédés le moins récemment.

5.2.4 Conclusion

Malgré sa simplicité, cette expérience nous a permis de remarquer une modification des performances du cache lors de changements de stratégies. Nous avons notamment pu mettre en évidence l'intérêt pour une stratégie de partir sur un état laissé par une autre stratégie. De plus, nous avons étudié les manières de gérer la

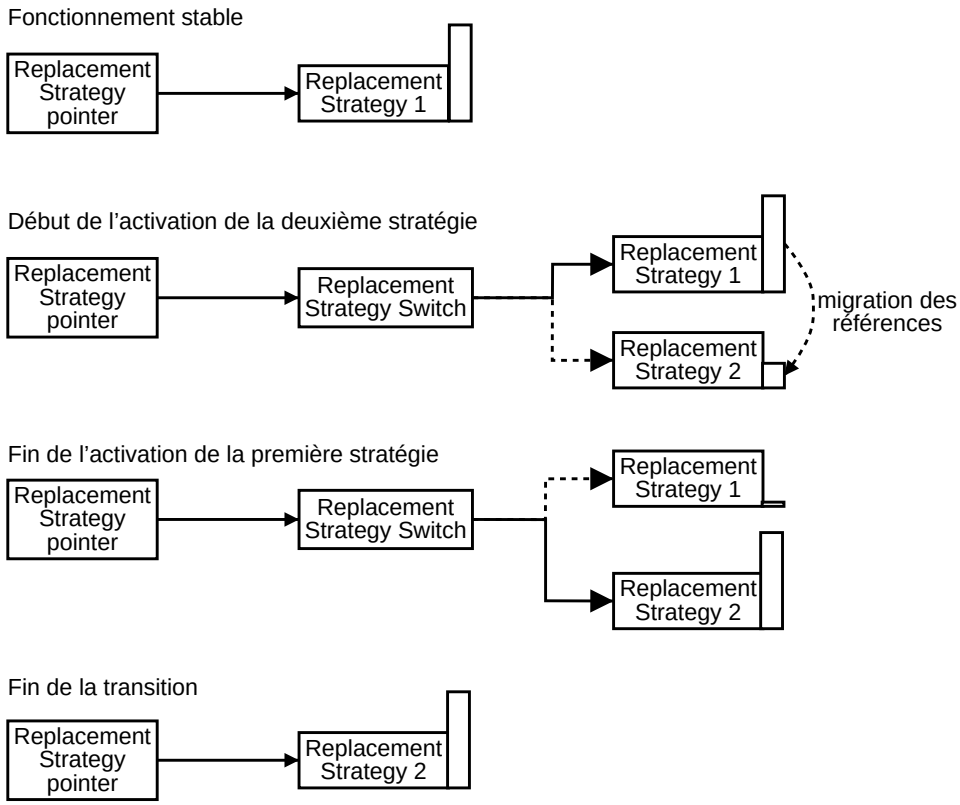

FIG. 5.6 — Mécanisme de transition

phase de transition entre deux algorithmes, tant au niveau du mécanisme lui-même qu'au niveau de la transmission d'informations sur l'état des stratégies.

5.3 Jigsaw

Jigsaw [Tea99, W3Ce] est une plate-forme de développement de services web développée par le consortium W3. Son objectif principal est de fournir une plate-forme expérimentale afin de valider les standards proposés par le consortium. Sa structure modulaire en fait une base de choix pour y développer des extensions. Nous allons dans un premier temps décrire la structure générale de Jigsaw, puis nous verrons la manière dont nous avons intégré un système de stockage adaptatif des données dans le système.

5.3.1 Structure

Pour Jigsaw, toute entité accessible à travers le serveur constitue une Ressource, qui est matérialisée par une instance d'un objet contenant les informations relatives

à l'entité considérée. Pour un fichier, ces informations sont le nom du fichier, sa taille, ses dates de modification et d'accès, etc. Les méthodes d'accès aux différentes ressources sont gérées par des ProtocolFrame, typiquement un HTTPFrame, que l'on associe à chaque ressource et qui offre un protocole particulier pour accéder aux données.

Un contrôle peut être effectué sur le résultat d'une requête en associant des filtres, héritant de la classe Filter, à l'objet Ressource ou ProtocolFrame adéquat. Les filtres, appliqués avant ou après l'accès aux données de la ressource, ont la capacité de modifier la requête ou le résultat de la requête produite, fournissant ainsi un mécanisme générique de manipulation des données.

On peut ainsi pratiquer une contrôle de l'accès à des données en appliquant un filtre d'authentification appelé GenericAuthFilter au ProtocolFrame associé à la donnée. Ce filtre est activé avant l'accès propre aux données et renvoie une réponse négative si l'accès est interdit. De manière similaire, le filtre GZIPFilter, invoqué après l'accès à la ressource, compresse le résultat de la requête avant de la renvoyer au client.

La figure 5.7 présente l'exemple de l'accès au contenu de la ressource `/archives/index.html`. Les répertoires `racine` et `archives` sont des ressources du type DirectoryResource. À chacune des ressources est attaché un HTTPFrame qui est lui-même relié à un filtre. Une requête pour la ressource `/archives/index.html` parcourt l'ensemble des filtres depuis celui de la racine jusqu'au filtre final. Une fois arrivé au niveau de la ressource finale, une réponse est élaborée, qui fera à son tour le chemin inverse de filtre en filtre avant d'être renvoyée au serveur.

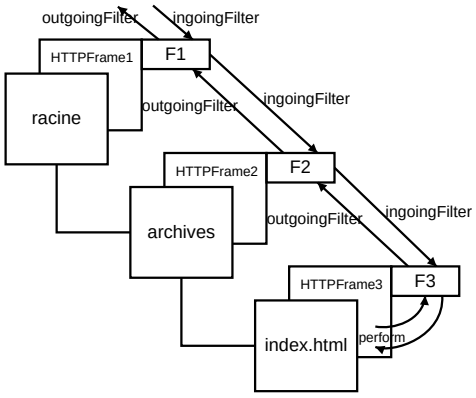

Fig. 5.7 — Accès à une ressource de Jigsaw

Un cache web peut être ainsi implémenté sous la forme d'un ProxyFrame que l'on attache à une ressource quelconque. Le ProxyFrame reçoit toutes les demandes de documents qui sont effectuées sur la ressource (qui doit être référencée comme point d'entrée du serveur de cache) et consulte la liste des documents dont une copie est conservée dans le cache sous la forme d'une ressource.

5.3.2 Stratégie de stockage

Le module de cache proposé par Jigsaw offre déjà une abstraction des données copiées, sous la forme d'une ressource EntityCachedResource. Toute ressource est cependant stockée sous la forme d'un fichier classique. Le placement des données est donc fixé à l'intérieur du cache et peut, comme nous l'avons souligné dans le paragraphe 1.7.9, souffrir d'une inadéquation des performances des systèmes de fichiers vis-à-vis des besoins du système de cache. Nous avons donc voulu, inspirés par les travaux effectués dans le projet ISTORE [BOK+99], introduire une stratégie de stockage plus flexible, permettant de changer dynamiquement le mode de stockage des données.

Implémentation

Nous avons identifié plusieurs stratégies de stockage possibles. La stratégie classique FileStorage consiste à stocker la copie de la ressource dans un fichier du système de fichiers. Nous avons introduit la stratégie MemoryStorage, qui stocke la ressource uniquement en mémoire. Cette stratégie est destinée à gérer des ressources de taille réduite et à courte durée de vie. De plus, nous avons introduit des stratégies hybrides MemoryThenDisk et DiskThenNetwork, afin de gérer les documents de taille plus importante. Leur principe est de stocker une fraction du document sur un support rapide et le reste sur un support plus lent mais moins coûteux. Ainsi, le temps de réaction lors d'une requête est diminué mais les ressources de stockage ne sont pas forcément pénalisées en cas d'abandon de transfert. La stratégie MemoryThenDisk ne présente pas d'intérêt dans le cas d'un système de cache fonctionnant au dessus d'un système d'exploitation, ce dernier prenant en charge ce type de fonctionnalité via le système de gestion de fichiers et le mécanisme de mémoire tampon. Cependant, certains constructeurs de caches [Ink] préfèrent utiliser un accès direct aux disques de stockage et gérer eux-mêmes le chargement en mémoire des données. La stratégie MemoryThenDisk peut alors présenter un intérêt. La stratégie DiskThenNetwork est plus intéressante. Si la connexion avec le serveur original du document est suffisamment rapide, il peut être intéressant de ne garder que le début du document dans le cache, et de rapatrier le reste via le réseau uniquement lorsqu'il est demandé. Cette technique permet également de gérer le cas des chargements interrompus en cours de transfert.

La mise en œuvre des différentes stratégies est effectuée en suivant le patron de conception *Strategy*, comme le présente la figure 5.8.

L'aspect de contrôle est isolé dans une classe Controller, qui implémente une méthode `getStorage` renvoyant une référence sur la stratégie de stockage la plus appropriée pour le document à stocker. La politique employée à ce niveau pour nos expériences a été un simple choix entre la stratégie FileStorage et la stratégie MemoryStorage, utilisant comme critère la taille de la ressource considérée. Nous avons donc mis en œuvre une adaptation *on action*, dont le processus d'adaptation a

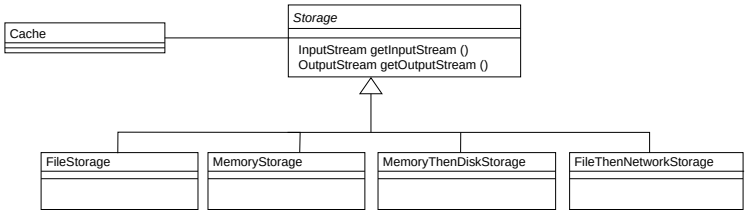

FIG. 5.8 — Stratégies de stockage

lieu lors de la première tentative de stockage. Lors des accès ultérieurs au document, la stratégie de stockage peut être modifiée : un document très populaire pourra passer de la stratégie FileStorage vers la stratégie MemoryStorage.

Nous avons de plus développé un filtre EventObserverFilter permettant la collecte des événements ayant lieu au niveau d'une ressource, qui fait partie de l'infrastructure nécessaire à la mise en place de la composante d'observation du système.

Nos mesures ne nous ont pas permis de mettre en évidence un changement notable des performances. Deux raisons peuvent expliquer ce constat : d'une part, l'architecture extrêmement modulaire de Jigsaw apporte déjà un coût relativement élevé (même s'il reste raisonnable [Tea98]) qui est peu modifié par les nouvelles indirections introduites. De plus, notre test a porté sur des stratégies de stockage disque et mémoire, qui ne présentent pas de grandes différences à cause du mécanisme de *buffering* du système de fichiers. Cependant, notre propos ici n'était pas d'étudier les performances mais le mécanisme de l'adaptation.

Travaux connexes

La modularisation des systèmes de cache n'est pas une caractéristique propre à Jigsaw. Le système *pluxy* [Ded98] est un *proxy* web dynamiquement extensible, permettant à un utilisateur de modifier dynamiquement certains de ses aspects. On y trouve notamment un système de cache fournissant deux types de fonctionnement, connecté et déconnecté. Le mode déconnecté utilise les données déjà présentes dans le cache sans chercher à en vérifier la validité sur le serveur original. Le passage du mode connecté au mode déconnecté s'effectue à la demande de l'utilisateur.

Des développement récents autour de Squid [Cha01] visent également à abstraire les stratégies de stockage du système, afin de pouvoir utiliser d'autres systèmes de stockage que le système de fichiers du système d'exploitation utilisé. L'organisation des politiques de remplacement a également été modifiée en conséquence, introduisant une architecture qui permettrait de mettre en œuvre un changement dynamique de stratégie.

5.3.3 Conclusion

Nous avons mis en œuvre les principes du patron de conception *Stratégie Autoadaptative* à l'intérieur de Jigsaw, conduisant à un cache fonctionnel. Cependant, les performances mesurées n'ont pas montré de différences significatives avec la version originale. Le développement d'une politique de contrôle adéquate devrait faire suite à ces travaux et valider le modèle.

5.4 Conclusion

Nous avons appliqué le patron de conception *Stratégie Autoadaptative* à deux catégories de systèmes, dans le but d'étudier les mécanismes inhérents au processus d'adaptation. Les systèmes de tri adaptatifs présentent un exemple simple d'adaptation *on action*, mais la pertinence du processus d'adaptation n'est pas évidente lorsque l'on met en parallèle le coût de l'adaptation, et particulièrement de la phase de caractérisation des données, et le coût du tri. Dans le domaine des caches web, nous avons étudié deux manières différentes d'introduire des mécanismes d'adaptation. La première, mise en œuvre dans un simulateur, présente une structure d'adaptation *on change* appliquée aux politiques de remplacement. Nous avons pu notamment étudier un exemple de mécanisme de transition. La troisième expérience porte sur le développement de stratégies de stockage adaptatives dans le cadre de Jigsaw, ayant recours dans un premier temps une adaptation *on action*. Ces réalisations nous ont permis d'acquérir une expérience dans le développement des systèmes adaptatifs, que nous avons voulu exprimer sous la forme du patron de conception *Stratégie Autoadaptative*. Par rapport à l'objectif de développement de caches web autoadaptatifs, nous avons donc pu développer l'aspect du mécanisme d'adaptation. Les étapes ultérieures de recherche concerneront l'aspect de contrôle et d'observation, avec la mise en œuvre de techniques de fouille de données (*data mining*) permettant de déterminer les conditions de fonctionnement du système.

Conclusion

Bilan

Le travail effectué s'inscrit dans la recherche de solutions permettant d'augmenter l'autonomie des systèmes informatiques dont la complexité croissante devient toujours plus difficile à gérer par les moyens classiques. Ainsi, l'administration des systèmes de caches web est de plus en plus ardue, à cause du grand nombre de réglages possibles et du caractère très variable de leur environnement d'exécution. L'autonomie des systèmes peut être atteinte par l'application du principe d'adaptation qui a pour objectif essentiel d'assurer la viabilité du système (informatique, biologique, etc) dans lequel il est mis en œuvre. Cette propriété de viabilité se traduit de diverses manières selon le système considéré.

Nous avons tout d'abord étudié les caractéristiques des systèmes de caches web afin de déterminer dans quelle mesure et de quelle manière le principe d'adaptation pouvait y être mis en œuvre. Nous avons constaté d'un côté qu'ils disposent d'un grand nombre de variantes possibles pour fournir un même service (stockage, remplacement, coopération, etc), accroissant par là même la difficulté de leur administration. Par ailleurs, nous avons mis en évidence le caractère extrêmement fluctuant de leurs conditions d'exécution. Ces deux facteurs font des systèmes de caches un sujet propice à la mise en œuvre de l'adaptation.

L'adaptation ne présente cependant pas une seule forme. Nous avons dans un premier temps clarifié le domaine de l'adaptation. Pour ce faire, nous sommes partis de son objectif essentiel que représente la viabilité pour voir de quelle manière il se décline suivant les domaines étudiés. Cette multiplicité de points de vue sur l'adaptation se traduit dans les recherches effectuées par une grande variété d'approches, dans lesquelles on peut toutefois trouver des propriétés communes (aspect temporel, multiplicité, observation). Nous avons donc présenté certains critères permettant de préciser le cadre d'application des différentes approches. Le premier critère identifie la structure du système soumise à l'adaptation (paramètre, comportement, architecture). Le second est la localisation de l'adaptation dans un schéma temporel à deux axes relatif au cycle de vie du logiciel, intégrant d'un côté le moment d'injection du comportement et de l'autre côté le moment où l'adaptation est effectuée. Sur la

base de ces critères, nous avons proposé une dénomination permettant de spécifier le mode d'adaptation que l'on considère. Nous avons appliqué cette dénomination à différentes approches que nous avons identifiées, qui sont désignées par différents termes : l'adaptabilité, qui exprime une capacité d'adaptation, est rendue tangible par la conception de systèmes adaptatifs. L'adaptabilité y est traduite par divers moyens relevant de la variabilité ou de la configurabilité du système.

À la lumière de cette classification, nous avons pu préciser le type d'adaptation le plus adapté au domaine des caches web que nous avons étudié précédemment : l'adaptation dynamique de comportement. Nous avons proposé le patron de conception *Stratégie Autoadaptative* qui définit une architecture commune aux différentes applications effectuant une adaptation dynamique de comportement, faisant abstraction des techniques de mise en œuvre utilisées. Les composants de cette structure sont étroitement liés aux différentes étapes du processus d'adaptation que nous décrivons. Ils isolent notamment l'aspect d'exécution du comportement de l'aspect de contrôle et soulignent l'importance de la prise en compte de la phase de transition entre deux stratégies. Nous avons étudié la manière dont le patron de conception *Stratégie Autoadaptative* permet de répondre aux besoins de deux types d'adaptation, *on action* et *on change*, puis nous avons précisé pour chacun des éléments les problématiques que l'on peut rencontrer dans leur mise en œuvre. Enfin, nous avons décrit la manière dont les deux types d'adaptation traités peuvent être combinés, ainsi que la capacité du patron de conception de s'appliquer de manière récursive à certains de ses éléments.

Nous avons ensuite étudié quelques systèmes autoadaptatifs à l'aune du modèle d'architecture proposé par le patron de conception *Stratégie Autoadaptative*, afin d'en vérifier la pertinence et de montrer une de ses utilisations possibles : l'analyse de systèmes existants. Nous avons tout d'abord analysé des applications adaptatives issues de domaines différents : le stockage de données, les systèmes d'exploitation, les protocoles de communication. Nous avons ensuite vu quels aspects du patron de conception sont pris en compte par des outils ou des techniques destinées à mettre en œuvre le modèle d'adaptation de comportement. Enfin, nous avons présenté les réalisations qui nous ont servi de support initial dans l'étude de ce sujet, et notamment décrit la mise en œuvre du mécanisme d'adaptation de stockage que nous avons effectuée dans le système de cache web Jigsaw.

Prolongements et perspectives

Nous pouvons à partir de ce travail considérer plusieurs perspectives de recherches.

La première perspective, immédiate, concerne la réalisation effective de l'adaptation au sein d'un cache web. Nous avons en effet ici traité uniquement du mécanisme de l'adaptation sans nous pencher en détail sur les autres aspects mis en jeu, notamment la prise de décision. Il s'agit donc de développer les techniques de contrôle

qui permettront de déterminer les différents types de fonctionnement des caches et les comportements les plus adéquats qui s'y appliquent. Cette étude touche donc particulièrement le domaine de l'analyse et de la fouille de données (*data mining*).

De plus, nous avons focalisé cette étude sur des systèmes ne présentant qu'une seule fonctionnalité offrant un comportement adaptatif. Cependant, et on a pu déjà le noter lorsque l'application récursive du patron de conception a été mentionnée dans le paragraphe 3.7, plusieurs stratégies autoadaptatives peuvent être mises en œuvre parallèlement dans le même système. En cas d'interaction entre des stratégies, il faut s'assurer de la cohérence des comportements modifiés. Ce problème couvre plusieurs aspects : le mécanisme de coordination des adaptations, la gestion de la stabilité du système résultant, ainsi que la manière de spécifier les dépendances entre les différentes stratégies. Pour ce dernier point, la définition de contrats de services [BJPW99] peut constituer une base solide.

En outre, la problématique de la transition, qui est symbolisée dans le patron de conception par le composant StateAdapter, peut être développée, notamment dans son aspect de transmission d'informations entre stratégies. Nous avons vu diverses techniques de mise en œuvre de cette transmission. Cependant, un cadre général de réflexion, sous la forme d'un patron de conception, nous semble adapté pour traiter de ce problème.

Le patron de conception *Stratégie Autoadaptative* n'est pas figé. Sa structure et son analyse peuvent évoluer suite à l'étude d'autres systèmes adaptatifs existants. Une suite naturelle est donc d'analyser d'autres applications adaptatives dans le but de raffiner le modèle. Parmi les domaines les plus propices à la continuation de cette étude, on peut citer en particulier celui des communications mobiles, de la robotique ou des systèmes multi-agents. Ces secteurs de recherche constituent en effet un terrain favorable à la mise en œuvre de l'adaptation par les besoins d'autonomie dans un environnement instable qu'ils expriment.

Enfin, on peut envisager la généralisation du modèle décrit dans le patron de conception *Stratégie Autoadaptative* à d'autres types d'adaptation parmi ceux que nous avons pu identifier dans notre étude de l'adaptation. Le modèle de processus de contrôle peut en effet se retrouver à différents niveaux, comme par exemple dans la phase de conception d'une application. Les problématiques identifiées relativement à la phase d'observation ou à celle de transition peuvent faire l'objet d'un traitement similaire à celui que nous avons effectué.

ANNEXE A

Le patron de conception *Stratégie Autoadaptative*

Nous présentons ici le patron de conception *Stratégie Autoadaptative* sous une forme condensée, suivant la structure utilisée dans [GHJV95].

A.1 Objectif

Le patron de conception *Stratégie Autoadaptative* permet à un système d'acquérir une forme d'autoadaptativité, n'exposant au client qu'une interface unique (*Facade*) accédant à la stratégie la plus adaptée aux conditions d'exécution courantes. Il suffit au client de fournir un accès aux informations d'exécution (environnement) qui permettent de choisir la meilleure stratégie.

A.2 Motivation

Considérons l'exemple d'un poste mobile devant accéder à des données situées sur des serveurs fixes [SKK+90, SA00]. Le dispositif mobile peut être déconnecté du réseau, et donc incapable d'accéder aux données, ou connecté au réseau et dans ce cas, la qualité de la connexion peut varier de manière importante, offrant un débit variable. Lorsqu'il est connecté avec un débit important (mode *fortement connecté*), le système mobile peut utiliser un mode de transmission standard, comme par exemple un protocole de transfert de fichiers classique. Lors de la déconnexion, une copie des données utilisées doit être créée localement afin que le fonctionnement de l'application ne soit pas perturbé. Un fonctionnement intermédiaire (mode *faiblement connecté*), en présence de perturbations, peut amener à dégrader le comportement de certaines fonctionnalités. On observe dans notre exemple trois comportements distincts : le mode fortement connecté, le mode faiblement connecté et le mode déconnecté. Le système mobile doit déterminer le mode de fonctionnalité courant, en

fondant sa décision sur l'obtention d'informations de son environnement important. Deux aspects interviennent dans ce problème : l'aspect fonctionnel (stockage des données) et l'aspect d'adaptation (détermination du mode de fonctionnement et choix du mécanisme adapté). Le patron de conception *Stratégie Autoadaptative* propose un mécanisme couvrant l'aspect d'adaptation.

A.3 Domaines d'application

Vous pouvez exploiter le patron de conception *Adaptive Strategy* lorsque :

- différentes versions d'un algorithme sont disponibles, et vous ne pouvez décider avant l'exécution quelle version est la plus adaptée ;
- vous créez un système autonome (ou agent) dont le comportement peut varier au cours du temps en fonction de conditions d'exécution internes ou externes.
- la méthode utilisée pour sélectionner un algorithme est bien définie, et le client ne devrait pas avoir à l'implémenter lui-même.

A.4 Structure

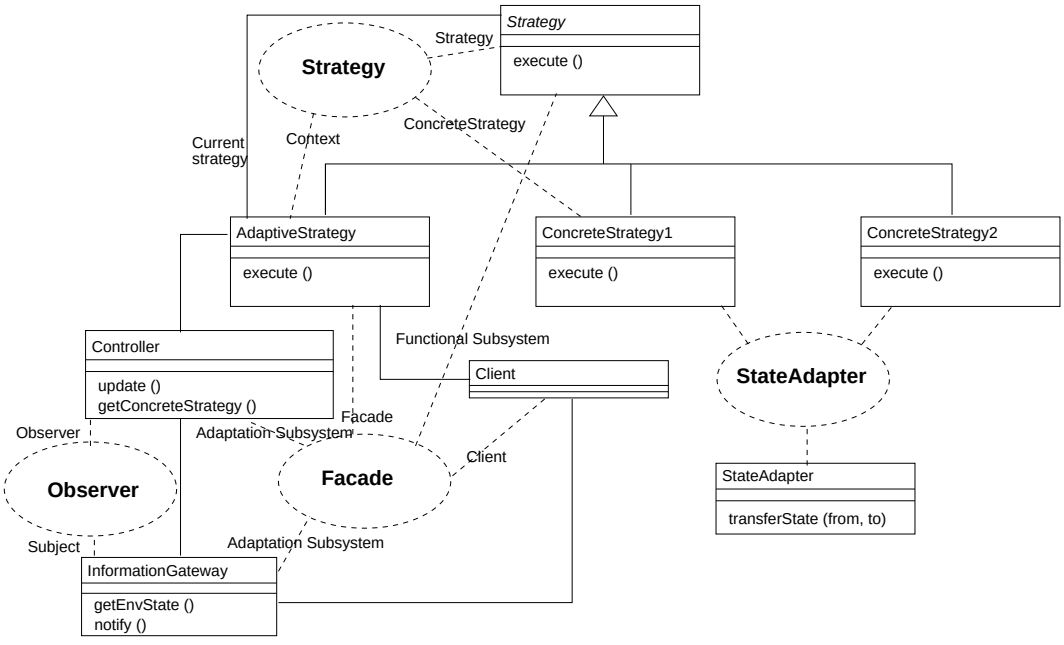

FIG. *A.1* — Structure générale du patron de conception

A.5 Participants

Client accède aux services de la stratégie via le composant AdaptiveStrategy, et fournit des informations sur ses conditions de fonctionnement via le composant InformationGateway.

Strategy déclare une interface commune aux différentes stratégies concrètes utilisables. Il représente la stratégie abstraite. Client utilise cette interface pour invoquer le service défini par un composant ConcreteStrategy.

ConcreteStrategy implémente une stratégie concrète, en respectant l'interface définie par Strategy. Chaque ConcreteStrategy peut comprendre notamment un état, selon la nature du service implémenté.

AdaptiveStrategy est le principal point d'accès pour les clients du système adaptatif, et fonctionne comme une *Facade* du système. Il fournit un point d'accès unique aux fonctionnalités des différentes stratégies.

Controller doit effectuer le choix de la meilleure stratégie, en utilisant l'information fournie par InformationGateway, mettant en œuvre le patron de conception *Observer*. Il est invoqué par le composant AdaptiveStrategy.

InformationGateway représente le canal par lequel le composant Controller obtient les informations nécessaires à sa prise de décision. Il fournit des méthodes d'accès aux paramètres d'environnement. Il peut également proposer des fonctionnalités actives, déclenchant des actions en cas de changement dans l'environnement, via des notifications aux autres composants.

StateAdapter gère si besoin est les transitions entre deux stratégies dans le cas où il est nécessaire d'échanger, en les transformant éventuellement, des informations sur leurs états respectifs.

On peut associer chacun des composants principaux du patron de conception à une étape du processus d'adaptation, comme le présente la figure A.2.

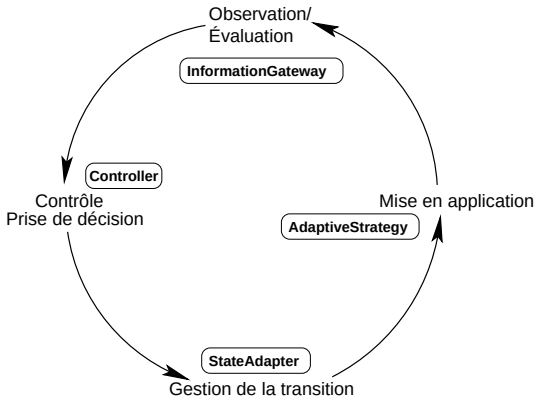

FIG. *A.2* — Le cycle de l'autoadaptativité et les composants du patron de conception *Stratégie Autoadaptative*

A.6 Collaborations

On peut distinguer deux types d'adaptation [BMN+00] : l'adaptation *on action* et l'adaptation *on change*. L'adaptation *on action* est effectuée lors de l'invocation d'une méthode de la stratégie. Ce type d'adaptation est utilisé lorsque les paramètres utilisés pour choisir la stratégie concrète font figurer des paramètres de calcul. L'adaptation *on change* est déclenchée par un changement dans l'environnement, typiquement relevé par une notification du composant InformationGateway, indépendamment des invocations des méthodes.

L'adaptation *on action* utilise le composant InformationGateway dans un mode passif, pour obtenir les informations pertinentes sur l'environnement au moment de l'invocation. Le Controller est interrogé par le composant AdaptiveStrategy afin de déterminer la stratégie la plus concrète au regard des conditions d'exécution.

L'adaptation *on change* est déclenchée par une notification du composant InformationGateway, qui informe le Controller des changements survenus. Ce dernier peut alors choisir un nouveau ConcreteStrategy si besoin est.

A.7 Conséquences

L'utilisation du patron de conception *Stratégie Autoadaptative* amène les propriétés suivantes :

- *Il masque des détails au client. Stratégie Autoadaptative* fournit une interface unique permettant d'invoquer le meilleur algorithme, sans que le client ait besoin de connaître toutes les alternatives ni la méthode de choix.
- *Il distingue le processus d'adaptation du processus d'exécution.* Le patron favorise une séparation plus nette entre ces deux aspects, conduisant à une plus grande clarté dans la compréhension des mécanismes et de leurs interactions.
- *Il rend la phase de transition explicite.* La phase de transition est souvent négligée dans les systèmes adaptatifs. Le composant StateAdapter réifie cet aspect important.
- *Il peut s'appliquer de manière récursive.* Il est possible d'appliquer le patron de conception à ses propres composants afin de profiter de ses propriétés à différents niveaux (observation, contrôle).
- *Il a un impact sur l'extensibilité.* L'addition d'une nouvelle stratégie est simplifiée par la réification effectuée. Cependant, le Controller doit connaître l'ensemble des stratégies disponibles ainsi que leurs caractéristiques respectives, afin de pouvoir choisir la plus appropriée.
- *Il a un impact sur les performances.* La mise en œuvre du patron *Stratégie Autoadaptative* peut entraîner des surcoûts dûs aux mécanismes utilisés.

A.8 Mise en œuvre

Les points clés relatifs à la mise en œuvre du patron de conception sont détaillés dans le paragraphe 3.5. Nous reprenons ici succinctement les enjeux mentionnés.

- Le composant InformationGateway peut proposer deux types de fonctionnement : un fonctionnement actif de notification du Controller en cas d'événement notable, et un fonctionnement plus passif où il est invoqué par le Controller pour obtenir les informations sur les conditions d'exécution. Outre cet aspect *immédiat*, il peut éventuellement gérer des informations d'historique sur les événements passés, ou bien fournir informations prospectives par le biais de simulations. Sa structure est souvent hiérarchique, les éléments de plus haut niveau effectuant une synthèse des informations provenant de la base.
- Le Controller représente la concrétisation de l'objectif de la politique d'adaptation. Parmi les mises en œuvre les plus courantes, l'automate à états fournit un mécanisme puissant, mais qui peut être complexe à mettre en œuvre. On peut également trouver une approche de simulation de différents scénarios.
- Le composant StateAdapter symbolise deux aspects : d'une part, il faut mettre en œuvre un mécanisme d'activation des stratégies (verrou global, redirection contrôlée, etc) permettant d'assurer la cohérence du système durant la phase de transition. D'autre part, il peut être nécessaire de transférer des informations d'une stratégie à une autre, éventuellement en les adaptant.
- Le patron *Stratégie Autoadaptative* peut s'appliquer de manière à ses propres composants InformationGateway ou Controller. Par exemple, la combinaison des deux types d'adaptation *on action/on change* peut être mise en œuvre par l'application du patron au Controller afin de fournir un Controller adaptatif *on change*. Le mécanisme original d'adaptation (*on action*) invoque alors le Controller le plus adapté aux conditions d'exécution.

A.9 Exemples d'utilisation

On peut trouver le patron de conception *Stratégie Autoadaptative* dans des domaines variés. Le système d'exploitation VINO [SS98] l'exploite pour modifier son fonctionnement en tenant compte de ses conditions d'exécution. Il peut effectuer des simulations de différents scénarios afin de déterminer le plus efficace.

Le système ISTORE [BOK+99] propose une infrastructure de stockage à base de briques élémentaires, qui peuvent être recombinées en fonction des demandes et des conditions d'exécution. On peut ainsi obtenir une tolérance aux pannes. Les fonctionnalités d'observation sont introduites au niveau matériel, et centralisées ensuite par le système.

Dans le domaine des réseaux sans-fils, l'application de prescription médicale en milieu hospitalier Charcot, développée dans le cadre du projet MolèNE [SA00], est capable de modifier son mode d'accès à une base de données en fonction de la qualité

de la connexion au réseau sans-fil.

Enfin, la plate-forme Jaws [HS99, Hu98] de conception de serveurs web permet d'adapter diverses politiques aux conditions d'exécution. Par exemple, la politique de gestion de la concurrence, qui détermine comment le serveur gère les différents fils de contrôle, est modifiable dynamiquement en cours d'exécution, suivant l'observation des ressources disponibles.

A.10 Patrons de conceptions connexes

- *Strategy*
- *Observer*
- *Facade*
- *State*

Ce patron de conception est principalement, tout comme *State*, une extension du patron *Strategy*.

Index

Bibliographie

[AACGJ01] H. Albin-Amiot, P. Cointe, Y.-G. Guéhéneuc and N. Jussien, Instantiating and Detecting Design Patterns : Putting Bits and Pieces Together, in *16th IEEE conference on Automated Software Engineering (ASE'01)*, 2001.

[ABE00] A. Andersen, G. S. Blair and F. Eliassen, OOPP : A Reflective Component-Based Middleware, in *Proceedings of NIK*, 2000.

[AC98] J. Almeida and P. Cao, Measuring Proxy Performance with the Wisconsin Proxy Benchmark, Technical Report 1373, Computer Sciences Dept, Univ. of Wisconsin-Madison, april 1998, `http://www.cs.wisc.edu/~cao/papers/cao-wpb/index.html`.

[ACD+98] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich and T. Jin, Evaluating Content Management Techniques for Web Proxy Caches, Technical report, HP Laboratories, Palo Alto, CA, 1998, `http://www.hpl.hp.com/techreports/98/HPL-98-173.html`.

[AFA97] G. Abdulla, E. A. Fox and M. Abrams, Shared User Behavior on the World Wide Web, in *WebNet97*, October 1997.

[Ale64] C. Alexander, *Notes on the Synthesis of Forms*, Harvard University Press, 1964.

[Ale77] C. Alexander, *A Pattern Language : Towns, Buildings, Constructions*, Oxford University Press, 1977.

[Ale79] C. Alexander, *The Timeless Way of Building*, Oxford University Press, 1979.

[ALF98] G. Abdulla, B. Liu and E. A. Fox, Searching the World-Wide Web : Implications from Studying Different User Behavior, in *WebNet98*, November 1998.

[App97] B. Appleton, *Patterns and Software : Essential Concepts and Terminology*, Object Magazine Online **3**(5) (1997).

[ASA+96] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams and E. A. Fox, Caching Proxies : Limitations and Potentials, in *Proc. 4th Int. WWW Conf.*, 1996.

[ASF98] M. Afonso, A. Santos and V. Freitas, QoS in Web Caching, in *3rd International WWW Caching Workshop*, 1998.

[AT98] M. Aksit and B. Tekinerdogan, Formalizing Adaptability Aspects (position paper), in *International Conference on Software Engineering, Aspect-Oriented Programming workshop*, 1998.

[AT99] M. Aksit and B. Tekinerdogan, Evaluating Architecture Implementation Alternatives based on Adaptability Concerns, in *Proceedings of 2nd IEEE International Symposium on Object-oriented Real-time distributed Computing, ISORC '99*, 1999.

[Aub99] O. Aubert, Towards fine-grained adaptivity in web caches, in *Work in progress, in Web Cache Workshop*, 1999.

[AW96] M. Abrams and S. Williams, *Complementing Surveying and Demographics with Automated Network Monitoring*, World Wide Web Journal **1**(3) (June 1996).

[AWA+95] M. Abrams, S. Williams, G. Abdulla, S. Patel, R. Ribler and E. A. Fox, Multimedia Traffic Analysis Using Chitra95, in *Proceedings of ACM Multimedia '95*, pages 267–276, november 1995.

[AWY99] C. C. Aggarwal, J. L. Wolf and P. S. Yu, *Caching on the World Wide Web*, IEEE Transactions on Knowledge and Data Engineering **11**(1), 95–107 (1999).

[BBM+97] M. Baentsch, L. Baum, G. Molter, S. Rothkugel and P. Sturm, *World Wide Web caching - the application level view of the internet*, IEEE Communications Magazine **35** (june 1997).

[BC87] K. Beck and W. Cunningham, Using Pattern Languages for Object-Oriented Programs, in *OOPSLA 1987 Workshop on the Specification and Design for Object-Oriented Programming*, 1987.

[BCF+99] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, Web caching and Zipf-like distributions : Evidence and implications, in *Proceedings of IEEE INFOCOM*, 1999.

[BDH+95] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber and M. F. Schwartz, *The Harvest information discovery and access system*, Computer Networks and ISDN Systems **28**(1–2), 119–125 (1995).

[Bee84] S. Beer, *The Viable System Model : Its Provenance, Development, Methodology and Pathology*, Journal of the Operational Research Society **1**(35), 7–25 (1984).

[BFPVG97] J.-C. Bolot, S. Fosse-Parisis and A. Vega-Garcia, Adaptive FEC-based error control for packet audio in the Internet, Technical report, INRIA, 1997, `http://www-sop.inria.fr/rodeo/fphone/biblio.html`.

[BFVY96] F. J. Budinsky, M. A. Finnie, J. M. Vlissides and P. S. Yu, *Automatic Code Generation from Design Patterns*, IBM Systems Journal **35**(2), 151–171 (1996).

[BGM+96] E. Borowsky, R. Golding, A. Merchant, E. Shriver, M. Spasojevic, and J. Wilkes, Eliminating storage headaches through self-management, in *1996 OSDI Symposium*, 1996.

[BH96] J.-C. Bolot and P. Hoschka, Performance Engineering of the World Wide Web : Application to Dimensioning and Cache Design, in *Fifth International World Wide Web Conference*, May 1996.

[BHB99] I. Bowman, R. Holt and N. V. Brewster, Linux as a Case Study : Its Extracted Software Architecture, in *Proceedings of ICSE99*, 1999.

[BJPW99] A. Beugnard, J.-M. Jezequel, N. Plouzeau and D. Watkins, *Making components contract aware*, IEEE Software (june 1999).

[BMN+00] P. Boinot, R. Marlet, J. Noyé, G. Muller and C. Consel, A Declarative Approach for Designing and Developing Adaptive Components, in *15th IEEE International Conference on Automated Software Engineering*, 2000.

[BMRaMS96] F. Buschmann, R. Meunier, H. Rohnert and P. S. an Michael Stal, *Pattern-Oriented Software Architecture - A System of Patterns*, Wiley and Sons Ltd., 1996.

[BN99] M. Braux and J. Noyé, Changement dynamique de comportement par composition de schémas de conception, in *Langages et Modèles à Objets LMO'99*, january 1999.

[BO00] G. Barish and K. Obraczka, *World Wide Web Caching : Trends and Techniques*, IEEE Communications Magazine Internet Technology Series (may 2000).

[BOK+99] A. Brown, D. Oppenheimer, K. Keeton, R. Thomas, J. Kubiatowicz and D. Patterson, ISTORE : Introspective Storage for Data-Intensive Network Services., in *Proceedings of the 7th Workshop on Hot Topics in Operating Systems (HotOS-VII), Rio Rico, Arizona*, march 1999.

[BP97] A. Baggio and G. Pierre, Oléron : supporting information sharing in large-scale mobile environments, in *Proceedings of the ERSADS '97 seminar*, Zinal, Switzerland, march 1997.

[BP99] J. F. Barnes and R. Pandey, Providing Dynamic and Customisable Caching Policies, in *Proceedings of the 2nd Symposium on Internet Technologies and Systems*, october 1999.

[BWKI98] S. Bagchi, K. Whisnant, Z. Kalbarczyk and R. K. Iyer, The Chameleon Infrastructure for Adaptive, Software Implemented Fault Tolerance, in *Proceedings of the 17 th IEEE Symposium on Reliable Distributed Systems*, pages 261–267, 1998.

[Cac99] CacheFlow, Network Cache Performance Measurements, Technical report, CacheFlow, 1999, `http://www.cacheflow.com/technology/wp/performance.html`.

[Cah96] V. Cahill, Flexibility in Object-Oriented Operating Systems : A Review, Technical Report TCD-CS-96-05, Department of Computer Science, Trinity College, Dublin, 1996, `citeseer.nj.nec.com/cahill96flexibility.html`.

[Cai95] R. Cailliau, A Little History of the World Wide Web, Technical report, W3C, 1995, `http://www.w3.org/History.html`.

[Car98] C. Carrez, Simulateur de caches web, Master's thesis, ENST Bretagne, 1998.

[CBC95] C. A. Cunha, A. Bestavros and M. E. Crovella, Characteristics of WWW Client-based Traces, Technical Report 95-010, Boston University Department of Computer Science, 1995.

[CDF+98] R. Caceres, F. Douglis, A. Feldmann, G. Glass and M. Rabinovich, Web Proxy Caching : The Devil is in the Details, in *1998 Workshop on Internet Server Performance*, 1998.

[CER] CERN – European Laboratory for Particle Physics, `http://www.cern.ch/`.

[CG00] C. Claveleira and C. Gross, Cache national du réseau Renater, 2000, `http://www.serveurs-nationaux.jussieu.fr/cache/`.

[Cha01] A. Chadd, Squid modio project, Technical report, Squid Cache Project, 2001, `http://squid.sourceforge.net/modio/info.html`.

[CI97] P. Cao and S. Irani, Cost-aware WWW proxy caching algorithms, in *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, pages 193–206, december 1997.

[Cis] Cisco, Cisco Cache Engine, `http://www.cisco.com/warp/public/cc/pd/cxsr/500/`.

[CMN83] S. Card, T. Moran and A. Newell, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, 1983.

[CMT01] I. Cooper, I. Melve and G. Tomlinson, *RFC3040 - Internet Web Replication and Caching Taxonomy*, jan 2001.

[Cor98] A. Cormack, Experiences with Installing and Running an Institutional Cache, in *3rd International WWW Caching Workshop*, June 1998.

[Cuna] W. Cunningham, Portland Pattern Repository, `http://c2.com/cgi/wiki?PortlandPatternRepository`.

[Cunb] W. Cunningham, Wiki Wiki Web, `http://c2.com/cgi-bin/wiki?WikiWikiWeb`.

[CZB98] P. Cao, J. Zhang and K. Beach, ActiveCache : Caching dynamic contents on the web, in *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, 1998.

[Dan98] P. Danzig, NetCache Architecture and Deployment, in *3rd International WWW Caching Workshop*, June 1998.

[DAP99] J. Dilley, M. Arlitt and S. Perret, Enhancement and Validation of Squid's Cache Replacement Policy, Technical report, HP Laboratories, Palo Alto, CA, 1999, `http://www.hpl.hp.com/personal/John_Dilley/caching/wcw.html`.

[Dav99a] B. Davison, Simultaneous Proxy Evaluation, in *Proceedings of the Fourth International Web Caching Workshop (WCW99)*, pages 170–178, april 1999.

[Dav99b] B. Davison, A survey of proxy cache evaluation techniques, in *Proceedings of 4th International Web Caching Workshop*, march 1999.

[Dav99c] B. D. Davison, Web Traffic Logs : An Imperfect Resource for Evaluation, in *Proceedings of the Ninth Annual Conference of the Internet Society (INET'99)*, June 1999.

[DB00] H. Duran and G. Blair, A Resource Management Framework for Adaptive Middleware, in *Proceedings of the 3rd IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC'2k)*, 2000.

[dC01] C. T. du CRU, Le projet Renater-Cache, Technical report, CRU, 2001, `http://www.cru.fr/renater-cache/`.

[Ded98] O. Dedieu, Pluxy : un proxy Web dynamiquement extensible, in *Proceedings of the 1998 NoTeRe colloquium*, oct 1998.

[DFKM97] F. Douglis, A. Feldmann, B. Krishnamurthy and J. Mogul, Rate of Change and other Metrics : a Live Study of the World Wide Web, Technical report, ATT Labs - Research, Digital Equipment Corporation, 1997.

[DHS93] P. Danzig, R. Hall and M. Schwartz, A case for caching file objects inside Inernetworks, in *Proc. SIGCOMM '93 (Computer and Communications Reviews 23(4)*, pages 239–248, October 1993.

[Dij74] E. Dijkstra, *Self stabilizing systems in spite of distributed control*, Commun. ACM **17**, 643–644 (1974).

[dJ97] A. de Jong, Report on the costs and benefits of operating caching services, Technical report, DESIRE project, 1997, `http://www.surfnet.nl/surfnet/projects/desire/deliver/WP4/D4-2.html`.

[DMF97] B. M. Duska, D. Marwood and M. J. Feeley, The Measured Access Characteristics of World-Wide-Web Client Proxy Caches, in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, dec 1997.

[DP96] A. Dingle and T. Partl, Web cache coherence, in *Fifth International WWW Conference*, 1996.

[dRS84] J. des Rivi'eres and B. C. Smith, The Implementation of Procedurally Reflective Languages, in *ACM Symposium on LISP and Functional Programming*, 1984.

[DSC+99] J. Dowling, T. Schafer, V. Cahill, P. Haraszti and B. Redmond, Using Reflection to Support Dynamic Adaptation of System Software : A Case Study Driven Evaluation, in *Proceedings of the OOPSLA '99 Workshop on ObjectOriented Reflection and Software Engineering*, 1999.

[Ede02] A. H. Eden, LePUS : A Visual Formalism for Object-Oriented Architectures, in *Proceedings of the 6th World Conference on Integrated Design and Process Technology*, 2002.

[FCAB98] L. Fan, P. Cao, J. Almeida and A. Z. Broder, Summary Cache : A Scalable Wide-Area Web Cache Sharing Protocol, Technical report, University of Wisconsin-Madison, 1998, `http://www.cs.wisc.edu/~cao/papers/summarycache.html`.

[FCD+99] A. Feldmann, R. Cáceres, F. Douglis, G. Glass and M. Rabinovich, Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments, in *Proceedings of IEEE Infocom'99*, pages 107–116, march 1999.

[FCLJ99] L. Fan, P. Cao, W. Lin and Q. Jacobson, Web Prefetching Between Low-Bandwidth Clients and Proxies : Potential and Performance, in *Measurement and Modeling of Computer Systems*, pages 178–187, 1999.

[FGM+97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, January 1997.

[FH92] N. Fenton and G. Hill, *Systems Construction and Analysis : A mathematical and logical Framework*, McGraw-Hill, 1992.

[FJ98] M. Frigo and S. G. Johnson, FFTW : An Adaptive Software Architecture for the FFT, in *ICASSP Conference Proceedings*, volume 3, pages 1381–1384, 1998.

[Flo96] S. Floyd, Adaptive Web Caching, in *2nd International Web caching workshop*, 1996.

[FPC00] H. E. Fargher, S. Pruitt and T. Cook, The Continuous Control and Adaptation of Systems using Online Architectures, in *The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000)*, edited by H. Arabnia, volume 2, pages 861–866, CSREA Press, June 2000.

[Gam91] E. Gamma, *Object-Oriented Software Development based on ET++ : Design Patterns, Class Library, Tools*, PhD thesis, University of Zürich, 1991.

[GH91] M. Gouda and T. Herman, Adaptive programming, in *ieeetse*, volume 17, pages 911–921, 1991.

[GHJV95] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns - Elements Of Reusable Object-Oriented Software*, Addison Wesley, 1995.

[GJ01] Y.-G. Guéhéneuc and N. Jussien, Using explanations for design-patterns identification, in *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, pages 57–64, 2001.

[Gla94] S. Glassman, A Caching Relay for the World Wide Web, in *Proceedings of the First International World Wide Web Conference*, 1994.

[GR83] A. Goldberg and D. Robson, *Smalltalk-80 : The Language and its Implementation*, Addison-Wesley Publishing Company, 1983.

[GR91] M. Gorlick and R. Razouk, Using Weaves for Software Construction and Analysis, in *Proceedings of the 13th International Conference on Software Engineering*, pages 23–34, 1991.

[Gra97] M. Gray, Internet Statistics : Web Growth, Internet Growth, Technical report, MIT, 1997, `http://www.mit.edu/people/mkgray/net/`.

[GRC97] S. Gadde, M. Rabinovich and J. Chase, Reduce, Reuse, Recycle : An Approach to Building Large Internet Caches, in *6th Workshop Hot Topics Operating Systems*, pages 93–98, IEEE, 05 1997.

[GSJ00] A. L. Guennec, G. Sunyé, and J.-M. Jézéquel, Precise modeling of design patterns, in *Proceedings of UML 2000*, edited by S. Verlag, volume 1939, pages 482–496, 2000.

[Har98] F. Haron, *Phase-based Adaptive Dynamic Load Balancing for Parallel Tree Computation*, PhD thesis, University of Leeds, 1998.

[Her01] C. Herring, The Pattern of the Viable System and its Language, in *Proceedings of KoalaPLoP 2001*, 2001.

[Hic01] M. Hicks, *Dynamic Software Updating*, PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 2001.

[HK00] C. Herring and S. Kaplan, The Viable System Architecture, in *Thirty-Fourth Hawaii International Conference on System Sciences (HICSS-34)*, 2000.

[HKC+99] C. K. Hess, F. Kon, R. H. Campbell, M. Román, D. Carvalho and L. Magalhães, Dynamic Resource Management for Smart Environments : The 2K Approach, in *Inter-agency Workshop on Smart Environments*, Atlanta, Georgia, July 25-26 1999, Georgia Institute of Technology.

[HMN01] M. Hicks, J. T. Moore and S. Nettles, Dynamic Software Updating, in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 13–23, ACM, June 2001.

[HMS99] I. Hadzic, W. S. Marcus and J. M. Smith, Policy and Mechanism in Adaptive Protocols, 1999, `http://www.cis.upenn.edu/~boosters/ms-cis-9903.ps`.

[Hoa61] C. A. R. Hoare, *Quicksort*, Computing Journal (1961).

[Hol92] J. H. Holland, *Adaptation in natural and artificial systems*, MIT Press, 1992.

[HS99] J. Hu and D. C. Schmidt, *Domain-Specific Application Frameworks : Frameworks Experience by Industry*, chapter JAWS : A Framework for High Performance Web Servers, Wiley and Sons, 1999.

[Hu98] J. Hu, The JAWS Adaptive Web Server, Technical report, Washington University of St Louis, 1998, `http://www.cs.wustl.edu/~jxh/research/`.

[Hun01] J. Hunter, *Filter code with Servlet 2.3 model*, JavaWorld (2001).

[iCY97] K. ichi Chinen and S. Yamaguchi, An interactive prefetching proxy server for improvement of WWW latency, in *Proceedings of the Seventh Annual Conference of the Internet Society*, 1997.

[Inf] Infolibria, Infolibria DynaCache, `http://www.infolibria.com/products/dynacache_overview.html`.

[Ink] Inktomi, Inktomi Content Networking Platform, `http://www.inktomi.com/products/cns/`.

[JGS93] N. Jones, C. Gomard, and P. Sestoft, *Partial Evaluation and Automatic Program Generation*, Prentice Hall International, 1993.

[Joh97] C. Johnson, What's the Web Worth ? The Impact of Retrieval Delays on the Value of Distributed Information, in *Proceedings of the Time and the Web Seminar*, 1997.

[Jr95] C. R. J. Jr, *Yet still more on the interaction of adaptive filtering, identification, and control*, IEEE Signal Processing Magazine **12**(2) (1995).

[JS97] P. Jain and D. C. Schmidt, Service Configurator : A Pattern for Dynamic Configuration and Reconfiguration of Communication Services, in *Proceedings of the 3rd Pattern Language of Programming Conference*, 1997.

[Jun] The Junkbuster Proxy, `http://internet.junkbuster.com/`.

[KC99] F. Kon and R. H. Campbell, Supporting Automatic Configuration of ComponentBased Distributed Systems, in *Proceedings of the 5th USENIX Conference on Object Oriented Technologies and Systems, COOTS'99*, may 1999.

[KCJ99] T. P. Kelly, Y. M. Chan and S. Jamin, Biased Replacment Policies for Web Caches : Differential Quality-of-Service and Aggregate User Value, in *Proceedings of 4th International Web Caching Workshop*, march 1999.

[KEWG96] M. Kaashoek, D. Engler, D. Wallach and G. Ganger, Server operating systems, in *SIGOPS European Workshop*, 1996.

[KLL+97] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine and D. Lewin, Consistent Hashing and Random Trees : Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web, in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 654–663, 1997.

[KLM+97a] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier and J. Irwin, Aspect-oriented programming, in *Proceedings of ECOOP'97—Object-Oriented Programming, 11th European Conference*, pages 220–242, 1997.

[KLM97b] T. M. Kroeger, D. D. E. Long and J. C. Mogul, Exploring the Bounds of Web Latency Reduction from Caching and Prefetching, in *Proceedings of the Symposium on Internet Technologies and Systems*, 1997.

[KMM97] T. M. Kroeger, J. Mogul and C. Maltzahn, Digital's Web proxy traces, Technical report, DEC, 1997, `ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html`.

[Knu73] D. Knuth, *The Art of Computer Programming, Volume 3 : Sorting and Searching*, Addison-Wesley, 1973.

[KR01] T. Kelly and D. Reeves, *Optimal Web Cache Sizing : Scalable Methods for Exact Solutions*, Computer Communications **24**, 173–173 (February 2001).

[KS98] P. Krishnan and B. Sugla, *Utility of co-operating Web proxy caches*, Computer Networks and ISDN Systems **30**(1–7), 195–203 (1998).

[KTP+96] T. Kimbrel, A. Tomkins, R. H. Paterson, B. Bershad, P. Cao, E. W. Felten, G. A. Gibson, A. R. Karlin and K. Li, A Trace-Driven Comparison of Algorithms for Parallel Prefetching and Caching, in *Second Symposium on Operating Systems Design and Implementation*, USENIX Association, 1996.

[KW98] B. Krishnamurthy and C. E. Wills, *Piggyback server invalidation for proxy cache coherency*, Computer Networks and ISDN Systems **30**(1–7), 185–193 (1998).

[KW99a] K. Kant and Y. Won, *Server capacity planning for Web traffic workload*, IEEE Transactions on Knowledge and Data Engineering **11**(5), 731–747 (september-october 1999).

[KW99b] B. Krishnamurthy and C. E. Wills, Proxy cache coherency and replacement–towards a more complete picture, in *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, june 1999.

[LA94] A. Luotonen and K. Altis, *World-Wide Web proxies*, Computer Networks and ISDN Systems **27**(2), 147–154 (1994).

[Lad99] R. Laddaga, *Creating robust software through self-adaptation*, IEEE Intelligent Systems , 26–29 (May 1999).

[LAJF98] B. Liu, G. Abdulla, T. Johnson and E. A. Fox, Web Response Time and Proxy Caching, in *WebNet98*, November 1998.

[LC97] C. Liu and P. Cao, Maintaining Strong Cache Consistency in the World-Wide Web, in *Proceedings of ICDCS'97*, pages 12–21, May 1997.

[Lie00] K. Lieberherr, Adaptive Programming - the Demeter project, Technical report, Northeastern University, 2000, `http://www.ccs.neu.edu/research/demeter/`.

[Liu98] B. Liu, Characterizing Web Response Time, Master's thesis, Virginia Tech, April 1998.

[LLB97] S. Lu, K.-W. Lee and V. Bhargavan, Adaptive Service in Mobile Computing Environments, in *Proceedings of the 5th International Workshop on Quality of Service (IWQOS'97)*, 1997.

[LPS97] K. J. Lieberherr and B. Patt-Shamir, Traversals of Object Structures : Specification and Efficient Implementation, Technical Report NU-CCS-97-15, College of Computer Science, Northeastern University, 1997.

[Mae87] P. Maes, *Concepts and Experiments in Computational Reflection*, PhD thesis, Vrije Universiteit Brussel, 1987.

[MAM98] C. D. Murta, V. Almeida and W. Meira, Jr, Analyzing Performance of Partitioned Caches for the WWW, in *Proceedings of the 3rd International WWW Caching Workshop*, jun 1998.

[Mar96] E. P. Markatos, Main Memory Caching of Web Documents, Technical report, Institute of Computer Science, Evangelos P. Markatos Computer Architecture and VLSI Systems Group Institute of Computer Science (ICS) Foundation for Research and Technology – Hellas (FORTH) P.O.Box 1385 Heraklio, Crete, GR-711-10 GREECE, 1996, `http://www5conf.inria.fr/fich_html/papers/P1/Overview.html`.

[MB97] M. Makpangou and E. Bérenguier, Relais : un protocole de maintien de cohérence de caches Web coopérants, in *Proceedings of the NoTeRe colloquium*, 1997.

[Mel97] I. Melve, Cost-Benefit Analysis, Technical report, DESIRE project, 1997, `http://www.uninett.no/prosjekt/desire/cost-benefit/cb-enkel.html`.

[Mic] Microsoft, Microsoft Proxy Server, `http://www.microsoft.com/proxy/`.

[MKPF99] E. P. Markatos, M. Katevenis, D. N. Pnevmatikatos and M. Flouris, Secondary Storage Management for Web Proxies, in *USENIX Symposium on Internet Technologies and Systems*, 1999.

[ML97] J. Mogul and P. Leach, RFC2227 : Simple Hit-Metering and Usage-Limiting for HTTP, 1997.

[MMF94] D. Mackenzie, R. McGrath and N. Friedman, Autoconf : Generating Automatic Configuration Scripts, Technical report, Free Software Foundation, 1994.

[MMV94] J. K. MacKie-Mason and H. R. Varian., Some economics of the Internet, in *10th Michigan Public Utility Conference at Western Michigan University*, page 37, February 1994.

[MP90] H. Massalin and C. Pu, *Fine-Grain Adaptive Scheduling using Feedback*, Computing Systems **3**(1), 139–173 (winter 1990).

[MRG99] C. Maltzahn, K. Richardson and D. Grunwald, Reducing the Disk I/O of Web Proxy Server Caches, in *Proceedings of the 1999 Usenix Technical Conference*, 1999.

[MRGM99] C. Maltzahn, K. Richardson, D. Grunwald and J. Martin, On Bandwidth Smoothing, in *Proceedings of 4th International Web Caching Workshop*, march 1999.

[MSL00] M. Mezini, L. Seiter and K. Lieberherr, Component Integration with Pluggable Composite Adapters, in *Software Architectures and Component Technology : The State of the Art in Research and Practice*, edited by M. Aksit, kluwer, 2000, University of Twente, The Netherlands.

[Mul97] G. Muller, Compte-rendu de la Journée CSAS'97, Technical report, IRISA, 1997, `http://compose.labri.fr/events/csas97/compte-rendu.html`.

[MW00] S. McClure and R. Wheeler, MOSIX : How Linux Clusters Solve Real World Problems, in *Proceedings of the 2000 USENIX Annual Technical Conference*, 2000.

[Net] Netscape, Netscape Proxy Server, `http://home.netscape.com/proxy/v3.5/index.html`.

[NFS00] D. Narayanan, J. Flinn and M. Satyanarayanan, Using history to improve mobile application adaptation, in *Proceedings of the 3rd IEEE Workshop on Mobile Computing Systems and Applications*, 2000.

[Not99] M. Nottingham, Optimizing Object Freshness Controls in Web Caches, in *Proceedings of the 4th International Web Caching Workshop*, 1999.

[NSN+97] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn and K. Walker, Agile Application-Aware Adaptation for Mobility, in *Proceedings of the 16th ACM Symposium on Operating System Principles*, october 1997.

[OCAV97] A. Ortega, F. Carignano, S. Ayer and M. Vetterli, Soft Caching : Web Cache Management Techniques For Images, in *IEEE Signal*

Processing Society 1997 Workshop on Multimedia Signal Processing, June 1997.

[OGT+99] P. Oreizy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum and A. Wolf, *An Architecture-Based Approach to Self-Adaptive Software*, IEEE Intelligent Systems , 54–62 (May 1999).

[Per] First International Workshop on Persistence and Java, `http://www.sun.com/research/forest/UK.Ac.Gla.Dcs.PJW1.pj1.html`.

[PfLSDC99] C. Placement and R. for Large-Scale Distributed Caches, *M. Korupolu and M. Dahlin*, IEEE Transactions on Knowledge and Data Engineering **11**(4), 549–562 (july-august 1999).

[Pie99] G. Pierre, *Architecture et dimensionnement d'infrastructures de caches Web pour Intranets décentralisés*, PhD thesis, Université d'Evry-val d'Essonne, Evry (France), June 1999.

[Pit97] J. Pitkow, In search of reliable usage data on the WWW, in *Proceedings of the Sixth International World Wide Web Conference*, april 1997.

[PKvST00] G. Pierre, I. Kuz, M. van Steen and A. S. Tanenbaum, Differentiated Strategies for Replicating Web documents, in *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, may 2000.

[Pla97] J. S. Plank, An overview of checkpointing in uniprocessor and distributed systems, focusing on implementation and performance, Technical report, University of Tennessee, 1997.

[PM96] V. N. Padmanabhan and J. C. Mogul, Using Predictive Prefetching to Improve World-Wide Web Latency, in *Proceedings of the SIGCOMM '96 conference*, 1996.

[PM00] S. Patarin and M. Makpangou, Monitoring Web Proxy Caches, Research Report RR-4023, INRIA, October 2000, `ftp://ftp.inria.fr/INRIA/publication/publi-ps-gz/RR/RR-4023.ps.gz`.

[Pov95] D. Povey, A Distributed Internet Cache, Technical report, University of Queensland, 1995, `http://www.psy.uq.edu.au/~dean/project/`.

[Rhi00] S. Rhine, Loadable Scheduler Modules on Linux, Technical report, Hewlett-Packard Management Solutions Lab, 2000.

[Riz98] L. Rizzo, Replacement policies for a proxy cache, Technical report, UCL-CS, 1998, `http://www.iet.unipi.it/~luigi/lrv98.ps.gz`.

[RKC99] M. Román, F. Kon and R. H. Campbell, Design and Implementation of Runtime Reflection in Communication Middleware : the dynamicTAO Case, in *Proceedings of the ICDCS'99 Workshop on Middleware*, June 1999.

[Rou97] A. Rousskov, Static Caching, in *2nd International Web caching workshop*, 1997.

[Rou98] A. Rousskov, Cache Digest, Technical report, NLANR, 1998, `http://squid.nlanr.net/Squid/CacheDigest/`.

[RS99] A. Rousskov and V. Soloviev, *A Performance Study of the Squid Proxy on HTTP/1.0*, WWW Journal (1999).

[RSB99] P. Rodriguez, C. Spanner and E. Biersack, Web caching architectures : hierarchical and distributed caching, in *Proceedings of WCW'99*, 1999.

[RSZ87] K. Ramamritham, J. Stankovic and W. Zhao, Meta-Level Control in Distributed Real-Time Systems, in *7th International Conference on Distributed Computing Systems*, pages 10–17, september 1987.

[RW98] A. Rousskov and D. Wessels, *Cache digests*, Computer Networks and ISDN Systems **30**(22–23), 2155–2168 (1998).

[RW99] A. Rousskov and D. Wessels, The First IRCache Web Cache Cache-off, Technical report, NLANR, 1999.

[RW00] A. Rousskov and D. Wessels, High Performance Benchmarking with Web Polygraph, Technical report, University of California, 2000, `http://www.web-polygraph.org/`.

[SA00] M. Segarra and F. André, A Framework for Dynamic Adaptation in Wireless Environments, in *Proc. of TOOLS Europe 2000*, june 2000.

[SESS94] M. Seltzer, Y. Endo, C. Small and K. A. Smith, An Introduction to the Architecture of the VINO Kernel, Technical Report TR-34-94, Harvard University, 1994.

[SG96] M. Seltzer and J. Gwertzman, World-Wide Web Cache Consistency, Technical report, Harvard University, 1996, `http://www.eecs.harvard.edu/~vino/web/usenix.196/`.

[SGHS01] E. Shriver, E. Gabber, L. Huang and C. Stein, Storage management for web proxies, in *Proceedings of the 2001 USENIX Annual Technical Conference*, 2001.

[Sha95] M. Shaw, *Beyond Objects : A Software Design Paradigm Based on Process Control*, ACM Software Engineering Notes **20**(1) (january 1995).

[SHA00] SHARP, *Super Proxy Script*, 2000.

[SHMP98] F. Sánchez, J. Hernández, J. Murillo and E. Pedraza, Run-Time Adaptability of Synchronization Policies in Concurrent Object Oriented Languages, in *ECOOP Workshop on Aspect-Oriented Programming*, 1998.

[SJJ+00] F. Sánchez, J.Hernández, J.M.Murillo, J.L.Herrero and R.Rodríguez, Adaptability of Object Distribution Protocols Using the Disguises Model Approach, in *2nd International Symposium on Distributed Objects and Applications*, 2000.

[SKK+90] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel and D. C. Steere, *Coda : A Highly Available File System for a Distributed Workstation Environment*, IEEE Transactions on Computers **39**(4), 447–459 (1990).

[SS98] M. Seltzer and C. Small, Self-monitoring and Self-adapting Operating Systems, in *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems*, 1998.

[SSV99] J. Shim, P. Scheuermann, and R. Vingralek, *Proxy Cache Algorithms : Design, Implementation and Performance*, IEEE Transactions on Knowledge and Data Engineering **11**(4), 549–562 (july-august 1999).

[Sta94] R. Stallman, *GNU Emacs Manual*, Free Software Foundation, 1994.

[SW49] C. Shannon and W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press : Urbana, 1949.

[Tay96] T. Taylor, *A Component- and Message-based Architectural Style for GUI Software*, IEEE Transactions on Software Engineering **22**(6), 390–406 (june 1996).

[TC98] M. Tatsubori and S. Chiba, Programming Support of Design Patterns with Compile-time Reflection, in *OOPSLA'98 Workshop on Reflective Programming in C++ and Java*, pages 56–60, 1998.

[TDVK98] R. Tewari, M. Dahlin, H. Vin and J. Kay, Beyond Hierarchies : Design Considerations for Distributed Caching on the Internet, Technical report, University of Texas at Austin, February 1998, `http://www.cs.utexas.edu/users/tewari/psfiles/tr98-04.ps`.

[Tea98] J. Team, Jigsaw Performance Evaluation, Technical report, W3C, 1998, `http://www.w3.org/Jigsaw/User/Introduction/performance.html`.

[Tea99] J. Team, Jigsaw 2.0 Internal Design, Technical report, W3C, 1999, `http://www.w3.org/Jigsaw/Doc/Programmer/design.html`.

[Tek96] B. Tekinerdogan, Modeling adaptability in object-oriented software development, in *ECOOP '96, Adaptability in Object-Oriented software development workshop*, 1996.

[TMW97] K. Thompson, G. Miller and R. Wilder, *Wide Area Internet Traffic Patterns and Characteristics*, IEEE Network **11**(6), 10–23 (nov/dec 1997).

[TS92] K. Thearling and S. Smith, An Improved Supercomputing Sorting Benchmark, in *Proceedings of Supercomputing '92*, IEEE Press : Los Alamitos, 1992.

[vGBS01] J. van Gurp, J. Bosch and M. Svahnberg, On the Notion of Variability in Software Product Lines, in *Proceedings of WICSA 2001*, august 2001.

[VR97] V. Valloppillil and K. W. Ross., Cache Array Routing Protocol v1.0, Technical report, Microsoft, 1997, `http://www.csm-usa.com/white/carpspec.htm`.

[vSHT99] M. van Steen, P. Homburg and A. S. Tanenbaum, *Globe : a wide area distributed system*, IEEE Concurrency **7**(1) (1999).

[VW] P. Vixie and D. Wessels, *RFC2756 (draft) - Hyper Text Caching Protocol*.

[W3Ca] W3C, Caching and Replication - the Problem, `http://www.w3.org/Propagation/Problem.html`.

[W3Cb] Consortium W3, `http://www.w3.org/`.

[W3Cc] W3C, eXtensible Markup Language, `http://www.w3.org/XML/`.

[W3Cd] W3C, HyperText Markup Language, `http://www.w3.org/MarkUp/`.

[W3Ce] W3C, Jigsaw - Web server platform, `http://www.w3.org/Jigsaw/`.

[W3Cf] W3C, Propagation, Caching and Replication on the Web, `http://www.w3.org/Propagation/`.

[W3Cg] W3C, Replication and Caching Position Statement, `http://www.w3.org/Propagation/Activity.html`.

[WA97] R. P. Wooster and M. Abrams, Proxy caching that estimates page load delays, in *WWW6 Proceedings*, pages 325–334, April 1997.

[Wan99] J. Wang, *A survey of web caching schemes for the Internet*, ACM Computer Communication Review , 36–46 (october 1999).

[WAS+96] S. Williams, M. Abrams, C. Standridge, G. Abdulla and E. A. Fox, Removal Policies in Network Caches for World Wide Web Documents, in *ACM SIGCOMM 96*, pages 293–305, August 1996.

[WC97] D. Wessels and K. Claffy, *RFC 2186 : Internet Cache Protocol (ICP), version 2*, September 1997.

[WC98] D. Wessels and K. Claffy, ICP and the Squid Web Cache, in *IEEE Journal on Selected Areas in Communication*, volume 16, pages 345–357, April 1998.

[WDR00] L. Wang, F. Douglis and M. Rabinovich, Forwarding Requests among Reverse Proxies, in *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, 2000.

[Wes98a] D. Wessels, Internet Cache Protocol, Technical report, NLANR, 1998, `http://ircache.nlanr.net/Cache/ICP/`.

[Wes98b] D. Wessels, Squid, Technical report, NLANR, 1998, `http://squid.nlanr.net/`.

[Wie48] N. Wiener, *Cybernetics or Control and Communication in the Animal and the Machine*, MIT Press, 1948.

[WM99] C. E. Wills and M. Mikhailov, Examining the cacheability of user-requested web resources, in *Proceedings of the 4th International Web Caching Workshop*, 1999.

[WMS98] B. WARFIELD, T. MUELLER and P. SEMBER, Monitoring the Performance of a Cache for Broadband Customers, in *INET'98 Conference Proceedings*, ISOC, 1998.

[Woo96] R. P. Wooster, *Optimizing Response Time, Rather than Hit Rates, of WWW Proxy Caches*, PhD thesis, Faculty of the Virginia Tech, December 1996.

[YJGMD96] T. W. Yan, M. Jacobsen, H. Garcia-Molina and U. Dayal, From User Access Patterns to Dynamic Hypertext Linking, in *Fifth International World Wide Web Conference*, May 1996.

[You94] N. Young, *The kserver dual and loose competitiveness for paging*, Algorithmica **11**(6) (1994).

Abstract

The unexpected growth of the WWW has highlighted new concerns. The capacity of the underlying infrastructure may have trouble handling the evergrowing amount of transferred data, which leads to performance issues from the point of view of the users as well as the ISPs.

Web caches are one of the answers to this problem. They replicate data, thus allowing faster transfer of documents and alleviating the load on networks and servers. However, they are complex systems running in a dynamic environment : they feature numerous algorithms for data storage, replacement or coherence. This complexity, which we may find in most complex computer systems like mobile computers for instance, makes their administration more difficult because of the great number of configuration parameters. Using self-adaptation mechanisms in these systems enhances their autonomy and makes them more easily manageable.

In order to reach this goal, we propose a classification of adaptation systems and define the different words and mechanisms involved. In the more precise scope of dynamic adaptation of behaviours, we propose the *Adaptive Strategy* Design Pattern which defines an architecture model of an adaptive application. It identifies the various components of the system and describes their interactions during the adaptation process. Analysis of existing adaptive systems as well as experiments on a web cache simulator and on a genuine web cache have been carried out to validate the proposed model.

Keywords : web caches, self-adaptive systems, design pattern

Résumé

La popularité sans cesse croissante du WWW incite à considérer de nouveaux enjeux qui n'étaient pas forcément d'actualité lors de la création de cet outil. La capacité des infrastructures de communication ne croît pas aussi rapidement que les besoins en termes de volume de données à transférer, ce qui pose des problèmes de performances tant pour les utilisateurs que pour les fournisseurs de services.

Des caches web ont été rapidement mis en place pour tenter d'obvier à cette difficulté. Ils permettent, par la réplication des données, d'accélérer le chargement des documents et de soulager la charge des réseaux et des serveurs. Cependant, le besoin de s'accommoder d'environnements aux besoins et caractéristiques très divers a conduit à l'élaboration de nombreux algorithmes dédiés aussi bien à la coopération des caches qu'au stockage des données, à leur remplacement ou à leur cohérence. Cette variété algorithmique, présente par ailleurs dans la plupart des systèmes informatiques complexes, comme les environnements sans fil par exemple, rend plus ardue leur administration du fait du grand nombre de réglages qu'elle entraîne. L'introduction de mécanismes d'adaptation automatique dans les systèmes augmente leur autonomie, et les rend plus facilement administrables.

Pour atteindre cet objectif, nous proposons une classification des systèmes d'adaptation permettant de préciser le vocabulaire et les mécanismes mis en jeu. Dans le cadre de l'adaptation dynamique de comportements, nous proposons le patron de conception *Stratégie Autoadaptative* qui définit un modèle d'architecture d'application autoadaptative. Il identifie les différents composants de tout système autoadaptatif et décrit leurs interactions lors du processus d'adaptation. Des analyses de systèmes adaptatifs existants ainsi que des expérimentations sur un simulateur et sur un cache web réel ont été conduites afin de vérifier son application.

Mots-clés : caches web, systèmes autoadaptatifs, patron de conception