Une approche adaptative basée sur la diversité pour la gestion des fautes dans les services Web

Hanane Abdeldjelil



Année 2013

Université Claude Bernard Lyon 1

Laboratoire d'InfoRmatique en Image et Systèmes d'information

École Doctorale Informatique et Mathématiques de Lyon

Thèse de l'Université de Lyon

Présentée en vue d'obtenir le grade de Docteur,
spécialité Informatique

par

Hanane Abdeldjelil

Une approche adaptative basée sur la diversité pour la gestion des fautes dans les services Web

---

Thèse soutenue le 20 Novembre 2013 devant le jury composé de :

M.	Franck MORVAN, Professeur, Université Paul Sabatier, Toulouse	Rapporteur
M.	Kokou YETONGNON, Professeur, Université de Bourgogne	Rapporteur
M.	Ladjel BELLATRECHE, Professeur, ENSMA Poitiers	Examinateur
M.	Claude GODART, Professeur, Université de Lorraine	Examinateur
M.	Djamal BENSLIMANE, Professeur, Université Lyon 1	Directeur de Thèse
Mme.	Noura FACI, Maître de conférences, Université Lyon 1	Co-Encadrante

Laboratoire d'InfoRmatique en Image et Systémes d'information
UMR 5205 CNRS - Lyon 1 - Bât. Nautibus
69622 Villeurbanne cedex - France

Remerciements

Je tiens à remercier toutes les personnes m'ayant accompagné tout au long de cette thèse.

Tout d'abord, je tiens à adresser mes chaleureux remerciements à mon directeur de thèse Djamal Benslimane pour son aide et ses précieux conseils ainsi que ses qualités humaines. Sa compétence et sa rigueur scientifique m'ont beaucoup appris. Je remercie également ma co-encadrante Noura Faci, pour toutes les réunions de travail, pour son aide, ses conseils et l'attention portée sur mes travaux. J'ai pris beaucoup de plaisir à travailler avec eux et je leur adresse ma très profonde gratitude.

Je souhaite adresser mes sincères remerciements aux personnes qui ont accepté la tâche délicate de rapporter ce mémoire et qui ont eu la patience de juger ce travail Frank Morvan, Professeur des Universités en Informatique à l'Université Paul Sabatier à Toulouse, et Koukou Yetongnon, Professeur des Universités en Informatique à l'Université de Bourgogneà Dijon .

Je souhaite remercier également les examinateurs et membres du jury, Ladjel Bellatrache, Professeur des Universités à l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique, et Claude Godart, Professeur, Université de Lorraine d'avoir accepté de participer au jury.

Je tiens tout particulièrement à remercier Zakaria Maamar, Professeur des universités en Informatique à l'université Zayed à Dubai pour sa collaboration et ses précieuses remarques sur mes travaux.

Je remercie spécialement la secrétaire de laboratoire LIRIS Madame Brigitte Gudayer pour sa spontanéité et son sourire qui a égayé nos journées.

Les années de la thèse n'auraient pas été possibles sans le soutien de certaines personnes formidables : Je remercie en premier lieu mes très chères amies de Tlemcen : Téma, Fatima Zohra, Sihem, Meriem et ma cousine Fatima pour tout ce qu'elles représentent pour moi.

Je remercie mes amis : Hind, Sofia, Soukaina, Noura, Lemya, Amel, Soumaya, Zohra, Ana, Dalila, Raafat et particulièrement ma très chère amie Sarah avec qui j'ai passé des moments inoubliables, je remercie également son mari Amine.

J'adresse toute mon affection à ma famille et en particulier à mes parents, à ma grand-mère, mes sœurs, Aicha et Amel et mes frères, Yacine et Sid-Ahmed. Malgré mon éloignement, votre confiance, encouragement et amour me guident tous les jours. C'est grâce à vous que je suis arrivée jusqu'ici. Je vous aime beaucoup.

Enfin, le dernier et pas des moindres, je remercier tout particulièrement mon fiancé Ridha pour son soutien quotidien, sa patience et son amour, je remercie également sa famille et tout particulièrement sa grand-mère.

Résumé

Les services Web tolérants aux fautes sont des composants avec une grande résilience aux défaillances qui résultent de différentes fautes imprévues, par exemple des bugs logiciels ou crash de machine. Comme il est impossible de prévoir l'apparition d'éventuelles fautes, de nombreuses stratégies consistent à dupliquer, d'une manière passive ou active, les composants critiques (eg. services Web) qui interagissent durant une exécution d'application distribuée (eg. composition). La capacité d'une application à continuer l'exécution en présence de défaillances de composants réfère à la Tolérance aux Fautes (TF). La duplication est la solution largement utilisée pour rendre les composants tolérants aux fautes. La TF peut être assurée à travers la réplication ou la diversité. Nous nous intéressons particulièrement dans cette thèse à la diversité, et nous montrons comment un ensemble de services Web sémantiquement équivalents qui fournissent la même fonctionnalité (eg. prévisions météo), mais qui l'implémentent différemment, collaborent pour rendre un service Web TF. Nous illustrons les limites de la réplication (présence de fautes répliquées), et proposons la diversité comme une solution alternative. En effet, la littérature a révélé un intérêt limité dans l'utilisation de la diversité pour rendre les services Web tolérants aux fautes.

Nous discutons d'abord la valeur ajoutée de la diversité pour assurer la TF dans l'exécution des services Web. Les services Web équivalents sont regroupés au sein d'un même espace virtuel appelé groupe de diversité (GD). S'appuyant sur cette similarité, nous concevons un GD d'un ensemble de services Web équivalents contrôlés par différents modèles d'exécution. Chaque modèle définit comment les services Web collaborent et interviennent lorsque l'un d'eux échoue. En plus de la notion du GD, des composants logiciels supplémentaires sont présentés à savoir : le manager du service composite (MCS), le manager de groupe de diversité (MGD) et le manager de service Web (MWS). Le premier composant invoque les GDs et soit il valide l'exécution globale soit il l'annule en cas d'échec. Le deuxième composant supervise l'exécution des services Web et les interactions entre eux au sein du GD. Enfin le dernier composant surveille l'exécution d'un service Web et signale l'échec ou le succès au MGD.

Une mise en place appropriée d'une stratégie de diversité nécessite également d'aborder différents détails comme le nombre nécessaire de services Web équivalents à impliquer, et les protocoles qui coordonnent l'exécution de ces services. Dans les approches de diversité existantes, tous les services Web disponibles dans un GD sont mis en action. Toutefois, cela n'est pas faisable tout le temps, la performance d'un service Web nécessite des ressources qui peuvent être limitées et parfois coûteuses. En outre, comme les valeurs de QoS (disponibilité et performance) nécessaires pour un GD varient d'une composition à une autre, seul un sous ensemble des services Web serait nécessaire. Ainsi, différentes configurations sont proposées en tenant compte de facteurs comme le degré de disponibilité et la déviation fonctionnelle à accepter. Dans ce travail, nous abordons des questions liées à combien de services Web équivalents sont nécessaires pour former un GD, quand et comment un service Web doit intervenir pour poursuivre l'exécution du GD en cas d'échec d'un service Web équivalent, et comment superviser le fonctionnement de tous ces services Web sans violer les exigences non fonctionnelles.

Finalement, nous proposons une architecture adaptative basée sur la diversité pour concevoir et déployer une composition des services Web fiable. Cette architecture adapte dynamiquement le service composite en surveillant en permanence le comportement des services Web composants dans les GDs, afin de réagir de manière appropriée aux fautes survenant au moment de l'exécution. Pour valider notre approche, nous avons conduit des expérimentations montrant l'utilisation de la diversité et son efficacité.

Mots-clefs : Sûreté de fonctionnement, Tolérance aux fautes, service Web, diversité.

Abstract

Fault Tolerant Web services are components with higher resilience to failures that result out of various unexpected faults for instance software bugs and machine crashes. Since it is impractical to predict the potential occurrence of a fault, a widely used strategy consists of duplicating, in a passive or active way, critical components (e.g., Web services) that interact during a distributed application execution (e.g., composition). The ability of this application to continue operation despite component failures is referred to as Fault Tolerance (FT). Duplication is usually put forward as a solution to make these components fault tolerant. It is achieved through either replication or diversity. In this thesis, we are particularly interested in diversity, and we show how semantically similar Web services, i.e., offer same functionality (e.g., Weather Forecast) but implement this functionality differently in terms of business logic and technical resources, collaborate together to make web services fault tolerant. We illustrate the limitations of replication (e.g., presence of replicated faults) and suggests diversity as an alternative solution. Our literature review revealed a limited interest in diversity for FT Web services.

We first discuss the value-added of diversity to support fault tolerant Web Services operation. Semantically similar Web services are gathering into the same virtual space referred to as diversity group (DG). Building upon this similarity, a DG consisting of semantically similar Web services is built and then controlled using different execution models. Each model defines how the Web services in the DG collaborate and step in when one of them fails to ensure operation continuity. Besides the DG, additional components are used including : composite manager (CM), group manager(GM), and Web service manager (WSM). The first component invokes the DGs and either resumes or aborts the overall execution in case of failures. The second component oversees Web services execution and the interactions between them. Finally, the last component monitors a Web service execution and reports either failure or success to the GM.

Setting up an appropriate diversity strategy also requires working out different details such as the necessary number of semantically Web services and protocols that coordinate the functioning of these Web services. In the existing diversity approaches, all available Web services in a DG are put into action. However this might not be doable all the time ; Web services performance requires resources that can be limited and sometimes costly. Moreover, as the QoS values (e.g., availability and performance) required to the GD can vary from one composition to another, only a subset of the Web services would be necessary. So, different configurations are offered taking into account factors like availability degree and functional deviation to accept. In this work, we raise questions related to how many similar Web services are necessary to form a DG, when and how a similar Web service needs to step in to take over from a failing Web service, and how to oversee the operation of all these Web services without violating non-functional requirementss.

We develop an adaptive diversity- based framework to make Web services FT that supports design and deployment of reliable composite Web services. This framework dynamically adapts the composite Web service by continuously monitoring the behavior of the component Web services in the DGs in order to react appropriately to faults occurring at runtime. It also offers several services like discovery/selection, failure detection, and recovery. Moreover, it provides developers with guidelines for designing and deploying FT Web services. To validate our approach, we run a set of experiments showing the use of diversity and its efficiency.

Keywords : Dependability, fault tolerant, Web service, diversity

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Contents

Ce chapitre est une introduction générale à cette thèse. Il décrit le contexte dans lequel s'inscrit ce travail et résume ensuite la problématique traitée et les contributions scientifiques proposées et mises en œuvre.

1.1 Contexte et motivations

Ces dernières décennies ont été marquées par le développement rapide des systèmes d'information distribués, et tout particulièrement par la démocratisation de l'accès à des données et à des applications via Internet.

Cette évolution du monde informatique a entraîné le développement de nouveaux paradigmes d'interaction entre applications. Cette thèse se situe dans le contexte des architectures orientées services (Service-Oriented Architecture, ou *SOA* [HB04]). Elle traite des mécanismes de tolérance aux fautes en vue de doter les applications à base de services des propriétés de sureté de fonctionnement.

L'architecture orientée service et plus particulièrement les services Web ont été mis en avant afin de permettre et de favoriser les interactions entre applications distantes, autonomes et hétérogènes aussi bien au niveau des plateformes qu'aux niveaux syntaxique et sémantique des données qu'elles peuvent manipuler.

Une architecture orientée services repose sur l'utilisation de fonctionnalités distribuées selon un modèle à trois types d'acteurs : les fournisseurs de services, les registres de services, et les consommateurs de services. Les fournisseurs de services déploient leurs services dans leurs propres plateformes et les décrivent selon une interface standard pour les rendre accessibles de l'extérieur. Les registres de services stockent les descriptions de services de l'ensemble des fournisseurs. Les consommateurs de services sélectionnent dans le registre de services les services qui répondent à leurs besoins, et les invoquent au travers de leurs interfaces. Un service [FB02] correspond à un composant logiciel fournissant une fonctionnalité et représente la brique de base d'une architecture orientée service. Il a vocation à être invoqué de façon dynamique dans diverses applications, et composé avec d'autres services pour répondre à des besoins complexes.

Les architectures orientées services sont mises en œuvre au travers essentiellement de protocoles et langages standards basés sur le langage XML tels que SOAP [1], WSDL [2], BPEL [3], WSCI [4], etc. De telles architectures offrent plusieurs avantages dont le couplage faible et l'interopérabilité standardisée des applications. Le couplage faible permet à une application d'invoquer des services autonomes et déployés sur des infrastructures différentes grâce à des protocoles de communication. Il n'y a nul besoin de disposer

1. Simple Object Access Protocol, http ://www.w3.org/TRsoap/
2. Web Service Description Language, http ://www.w3.org/TR/wsdl
3. Business Process Execution language, http ://bpel.xml.org/
4. Web Service Choreography Interface, http ://www.w3.org/TR/wsci

d'un système de middleware réparti commun comme c'est dans le cas des approches orientées objets. L'interopérabilité permet à une application client d'interagir avec un service distant malgré l'hétérogénéité des plateformes et des langages dans lesquels les services ont été développés.

Les applications critiques, orientées services ou pas, (e.g., domaines militaire et du contrôle aérien) peuvent nécessiter une haute sûreté de fonctionnement. Celle-ci est définie comme la caractéristique qui vise à pouvoir placer une confiance justifiée dans les services offerts par une application. Il est tout d'abord important de souligner la différence entre les concepts de défaillance, erreur et faute. Une défaillance se produit lorsque le service délivré dévie de ce qui est attendu. La partie du système susceptible d'avoir causée la défaillance est appelée erreur. Une faute est la cause adjugée ou supposée d'une erreur.

La sûreté de fonctionnement [ALRL04] d'une application est définie selon plusieurs dimensions : la disponibilité, la fiabilité, la sécurité, et la confidentialité. Dans ce travail de thèse, nous nous limiterons aux deux premières dimensions. La disponibilité d'une application correspond à sa capacité d'être atteint pour délivrer des résultats. La fiabilité d'une application correspond à sa capacité de continuer son fonctionnement en présence de fautes. La sûreté de fonctionnement est généralement accomplie par différentes méthodes : (1) la prévention de fautes pour éviter l'introduction de fautes aussi bien au niveau logiciel que matériel, (2) l'élimination de fautes via la maintenance corrective, (3) la prévision de fautes en évaluant le comportement de l'application en présence de fautes, et (4) la tolérance aux fautes en évitant la défaillance de l'application par des techniques de détection et de recouvrement d'erreurs. Comme il est pratiquement impossible de prévoir et d'éviter toutes les fautes, nous nous sommes intéressés aux aspects de tolérance aux fautes des applications à base de services.

Ces dernières, en raison de leur architecture et leur support de déploiement, sont particulièrement sujettes à des fautes. Il est naturellement envisageable que la défaillance au sein d'un de leurs composants non convenablement traitée peut se propager et peut par conséquent perturber le service fourni par l'application. De plus, chaque service Web participant à une application donnée introduit un potentiel point de défaillance. La conception et le déploiement de services tolérants aux fautes constituent donc notre principale motivation en vue d'aboutir à des applications à haute sûreté de fonctionnement.

1.2 Problématique

Assurer la disponibilité et la fiabilité des applications à base de services constitue un réel challenge. La tolérance aux fautes des services, atomiques ou composites, constitue une des principales approches de sûreté de fonctionnement. La stratégie de tolérance

aux fautes la plus largement répandue est basée sur la duplication. Celle-ci peut être réalisée soit par la réplication, les répliques sont des copies exactes du service d'origine, soit par la diversité [Avi95] qui consiste à recourir à des services différents mais fonctionnellement similaires.

La réplication d'un service est simple à mettre en œuvre mais malheureusement elle ne permet pas de faire face à certaines fautes inhérentes au service lui-même. Ces fautes se retrouveront naturellement dans toutes les répliques du service concerné. La diversité présente un avantage majeur par rapport à la réplication due à l'absence des fautes répliquées. Cependant, contrairement à la réplication où les différents protocoles de réplication (e.g., active et passive) sont bien définis et formalisés, très peu de travaux ont été consacrés aux mécanismes de tolérance aux fautes à base de diversité dans les des applications à base de services.

Ce travail de thèse adopte donc la notion de diversité dans le contexte d'architectures orientées services et étudie les questions suivantes pour les besoins de sûreté de fonctionnement :

- **Q1**. Comment concevoir et superviser le fonctionnement d'une application à base de services tolérante aux fautes basée sur la technique de diversité.
- **Q2**. Comment adapter les protocoles de réplication classique au cas de la diversité.
- **Q3**. Comment créer et configurer la diversité de services pour une meilleure sûreté de fonctionnement des applications à base de services. Plus précisément, il s'agit de déterminer le nombre n de services équivalents nécessaires pour tolérer un nombre f de fautes.
- **Q4**. Comment sélectionner les n meilleurs services dans un groupe de diversité de sorte à répondre aux besoins de la tolérance aux fautes.

1.3 Contributions

Les contributions de cette thèse se présentent comme suit :

- **C1. Définition d'une architecture multi niveaux pour la tolérance aux fautes des services**. Une architecture *multi* niveaux pour la conception et l'exécution de services tolérants aux fautes est proposée. Les fautes considérées sont de type crash et Byzantine. Le premier niveau permet de décrire les applications à base de services en termes de services abstraits. Le deuxième niveau permet la concrétisation des services abstraits au travers des groupes de diversité. Un groupe de diversité représente un ensemble de services similaires capables de répondre à la fonctionnalité d'un même service abstrait. La duplication des services sera réalisée uniquement via les éléments d'un même groupe de diversité. Le troisième niveau correspond aux services concrets mis à la disposition des applications. L'architecture propo-

sée permet aussi d'attacher des propriétés de criticité aux services Web abstraits pour définir la sévérité d'éventuelles fautes. Trois niveaux de criticité sont définis : critique quand la faute n'est en aucun cas acceptable, semi-critique quand des résultats partiels sont tolérés, et non critique quand la faute est totalement acceptée. A chaque niveau de l'architecture est associé un type de composant logiciel appelé manager pour surveiller et gérer les services concrets, les groupes de diversités et la composition de services. Le comportement de chaque manager proposé est décrit sous forme d'automates à états finis.

- **C2. Définition de protocoles de duplication adaptés à la diversité**. Deux protocoles de duplication sont définis et formalisés pour l'exécution tolérante aux fautes des services. Le protocole séquentiel permet le recouvrement de fautes via des exécutions séquentielles d'un sous-ensemble de services d'un même groupe de diversité. Le protocole parallèle réalise le recouvrement de fautes via une exécution parallèle de plusieurs éléments d'un même groupe de diversité. Un algorithme de vote est aussi proposé pour le cas du protocole parallèle. Il permet de choisir le bon résultat dans le cas de plusieurs réponses équivalentes de services d'un même groupe de diversité.
- **C3. Configuration des groupes de diversité**. Il s'agit essentiellement d'estimer pour un groupe de diversité le nombre de services similaires nécessaires pour l'exécution d'un service abstrait. Tout d'abord un algorithme pour le calcul du degré de redondance n exigé dans un groupe de diversité est proposé. Cet algorithme tient compte du protocole de recouvrement utilisé. Ainsi, dans le cas du protocole parallèle, il permet de déterminer le degré approprié de redondance à utiliser pour le prochain tour de vote tenant compte des fautes observées dans les précédents tours. Dans le cas du protocole séquentiel, il s'agit de réserver n services et définir leur ordonnancement.
- **C4. Sélection de meilleurs services dans un groupe de diversité**. Une fois un groupe de diversité est configuré pour fonctionner avec n services similaires, il s'agit alors d'identifier les n meilleurs services. Des algorithmes du sélection des n meilleurs services au sein d'un groupe de diversité sont proposés. Ils sont basés sur l'historique d'exécution de services Web, les exigences des clients et les changements de contexte en termes de variation du nombre de fautes détectées (i.e., augmentation ou diminution).
- **C5. Implémentation et expérimentation des concepts proposés**. Une mise en œuvre des protocoles de tolérance aux fautes a été réalisée. Les expérimentations montrent que la stratégie de sélection basée sur l'historique d'exécution (Log) améliore l'efficacité du groupe de diversité.

1.4 Organisation du manuscrit

Le reste du document est constitué des chapitres suivants :

- Le chapitre **2** décrit dans un premier temps les notions fondamentales relatives aux domaines des architectures orientées services et de la tolérance aux fautes. Il présente ensuite les principales approches de la sûreté de fonctionnement dans les Services Web. Finalement, nous nous positionnons par rapport aux approches proposées de tolérance aux fautes pour justifier les différents problèmes que nous abordons dans cette thèse.
- Le chapitre **3** présente notre architecture multi-niveaux d'applications à base de services pour la gestion de fautes. Elle est basée sur la diversité de services. Un modèle de fautes est identifié, et des propriétés de criticité sont définies et attribuées à la fonctionnalité implémentée par les services Web concrets. Les différents managers associés aux niveaux de l'architecture sont proposés et décrits en termes d'automates à états finis. Des protocoles séquentiel et parallèle sont proposés pour les groupes de diversité comme stratégies de recouvrement de fautes.
- Le chapitre **4** s'intéresse au problème de la configuration automatique des groupes de diversité. Une méthode de calcul du degré de redondance des services est proposée pour les protocoles séquentiel et parallèle. Ce chapitre présente ensuite des algorithmes de sélection des meilleurs services d'un groupe de diversité peuvant assurer la redondance précédemment calculée.
- Le chapitre **5** présente l'architecture logicielle qui implémente l'approche de tolérance aux fautes proposée dans le chapitre 3. Il implémente ensuite les algorithmes de sélection de services dans un groupe de diversité. Enfin, il présente quelques résultats d'expérimentation de l'approche proposée.
- Le chapitre 6 termine ce mémoire par une conclusion qui rappelle les contributions scientifiques de notre travail, et dresse quelques perspectives de recherche en liaison avec le sujet traité.

Chapitre 2

Concepts fondamentaux et État de l'art

Contents

Ce chapitre a pour objectif de présenter le cadre de départ de nos travaux. Dans un premier temps, nous introduisons les concepts de base des services Web et de la sûreté de fonctionnement. Nous présentons ensuite l'état de l'art des travaux réalisés sur la tolérance aux fautes dans les Services Web. Enfin, nous discutons les différentes approches pour se positionner par rapport aux solutions existantes.

2.1 Généralités

2.1.1 Architecture Orientée-Service et Services Web

L'Architecture Orientée-Service (*en anglais*, Service-Oriented Architecture (SOA)) est un paradigme architectural présentant un intérêt particulier pour les technologies de l'information et le domaine de l'entreprise. Le Consortium World Wide Web (W3C), chargé du développement des standards sur le Web et leur évolutivité, donne la définition suivante de SOA : "*collections structurées de composants logiciels, appelés services, à invoquer avec des descriptions d'interface pouvant être publiées et découvertes*" [HB04].

2.1.1.1 Principes SOA

Les composants de base de SOA sont : le fournisseur de service, le registre de service, et le client de service. Les fournisseurs de service sont des applications logicielles fournissant un service au client. Les fournisseurs publient une description des services offerts via un registre de services et les clients consomment ces services. Une application orientée-service peut agir comme fournisseur et consommateur de service à la fois. Les clients doivent être en mesure de trouver la description des services dont ils ont besoin et se lier à eux. Dans [Erl07], T. Erl propose plusieurs principes de conception sous-jacents à SOA comme suit :

- *Contrat de service standardisé* : Le contrat de service définit un accord entre le fournisseur et le consommateur, composé des éléments suivants : i) une représentation technique du service (i.e., son interface) comme les opérations, leurs paramètres d'entrée et de sortie et les contraintes sur les entrées ; ii) une description informelle des opérations sous formes de règles et contraintes d'utilisation du service (e.g., volume des données échangées) ; iii) un niveau de service (QoS et SLA) précisant les engagements du service (e.g., temps de réponse maximum attendu, plages horaires d'accessibilité, le temps de reprise après interruption, les procédures mises en œuvre en cas de panne, les procédures de prise en charge du support).
- *Faible couplage entre les services* : Les services maintiennent une relation minimisant les dépendances. Ils n'ont plus besoin d'un système de middleware réparti commun pour communiquer, mais seulement de **protocoles et technologies de**

communication interopérables sur Internet.
- *Abstraction de service* : En dehors des différentes descriptions dans le contrat de service, les services cachent leur logique au monde extérieur. Le contrat de service ne doit contenir que les informations essentielles à son invocation.
- *Services réutilisables* : La logique est divisée en différents services avec comme objectif de promouvoir la réutilisation. Les services peuvent être ainsi partagés parmi différents domaines.
- *Services autonomes* : Les services contrôlent la logique d'exécution qu'ils encapsulent. Plus ce contrôle est fort, plus l'exécution d'un service est prédictible.
- *Services dépourvus d'état* : Les services minimisent la consommation de ressources en déléguant la gestion des informations d'état quand cela est nécessaire.
- *Services découvrables* : La description des services est complétée par un ensemble de méta-données permettant leur découverte et leur interprétation de manière efficace et appropriée.
- *Services composables* : Les services sont conçus de manière à participer à des compositions de services. Les services peuvent être orchestrés pour améliorer l'agilité du processus métier implémenté.

2.1.1.2 Services Web

Les services Web sont l'une des technologies implémentant les principes SOA. Un service Web est une partie d'application faiblement couplée, indépendante, et mise à disposition sur le Web pouvant être décrite, publiée, découverte, coordonnée (ou composée) et configurée en utilisant des artefacts (e.g., XML [toE00]) pour une simplicité de développement d'applications hétérogènes dans des environnements distribués. Les services Web permettent à des applications de travailler ensemble à travers des protocoles Internet standards (e.g., HTTP - Hypertext Transfer Protocol [Con]) afin d'automatiser des opérations métier sans l'intervention humaine [FB02, CS01]. Dans les services Web, les clients et les services sont des systèmes à part entière et indépendants des uns des autres. Les services sont normalement autonomes, développés et déployés par différents fournisseurs de service.

Décrire un service Web revient à définir l'ensemble des fonctionnalités sous forme d'opérations offertes par le service (i.e., son interface) à l'aide de langages de description tel que *WSDL* (Web Service Definition Language [Cha02]) ou de ressources identifiables *URI* (Uniform Resource Identifier). Les services Web communiquent entre eux via des protocoles standard de communication tels que *SOAP* (Simple Object Access Protocol [SOA02]) reposant sur le formalisme *XML* pour le format des messages échangés ou *REST* (Representational state transfer [Fie])

Les clients peuvent accéder et localiser les services Web dans un annuaire tel que *UDDI* (Universal Description Discovery and Integration) [OAS04]. Un annuaire *UDDI* permet de localiser sur le réseau, un service Web accessible par l'intermédiaire du pro-

tocole *SOAP*.

La figure 2.1 montre les différentes étapes nécessaires à l'invocation d'un service Web :

Publication. Les fournisseurs mettent à disposition des services sur le Web et enregistrent la description de ces services dans un annuaire (étape 1).

Découverte. Le consommateur à la recherche d'un service répondant à ses besoins doit découvrir un service de *matching* parmi tous les fournisseurs. Il doit être en mesure de décrire clairement ses besoins (i.e., sa requête). La description du service désiré est comparée avec celle du service proposé. Si les deux services s'appartient, le service requis a été découvert avec succès. (étape 2)

Composition. Il est possible qu'aucun *matching* n'existe avec les besoins du consommateur. Dans ce cas, il est possible de composer des services existants pour répondre à ces besoins. La composition de service décrit la combinaison de deux ou plusieurs services en un service plus complexe. Il existe deux approches de composition de services décrites comme suit :

- *Chorégraphie*. Elle décrit la collaboration de services définie par un ensemble de règles d'interaction (ou protocoles) parmi les services sans aucune entité centrale de contrôle. Un langage largement utilisé est le Web Service Choreography Interface (WSCI) [HXW+07].
- *Orchestration*. Elle décrit la collaboration de services contrôlée par un composant central, appelé moteur de la composition. Ce moteur connait les règles pour les composer [BCP+]. BPEL (acronyme de "'Business Process Execution Language"') utilise cette approche [Jur06].

Binding. Après la découverte du service approprié délivrant la fonctionnalité souhaité, le service consommateur se lie à ce service pour l'exécution. A ce stade, des paramètres de sécurité (e.g., authentification, autorisation) doivent être configurés de part et d'autre. (étape 3)

Execution. Une fois le binding effectué, le service ou la composition de services peuvent être exécuté(e). Les paramètres d'entrée sont transmis au fournisseur de service et ceux de sortie sont renvoyés au consommateur. (étape 4)

Depuis leur apparition, les services Web sont de plus en plus demandés surtout dans le développement d'applications critiques relatives aux domaines tels que le contrôle aérien et la défense. Cependant ces applications peuvent être sujettes à des défaillances pouvant ainsi provoquer de réelles catastrophes humaines et/ou financières. Ainsi, assurer un fonctionnement sûr des services Web en dépit de l'occurrence de fautes devient l'une des questions les plus cruciales à laquelle nous nous intéressons dans cette thèse.

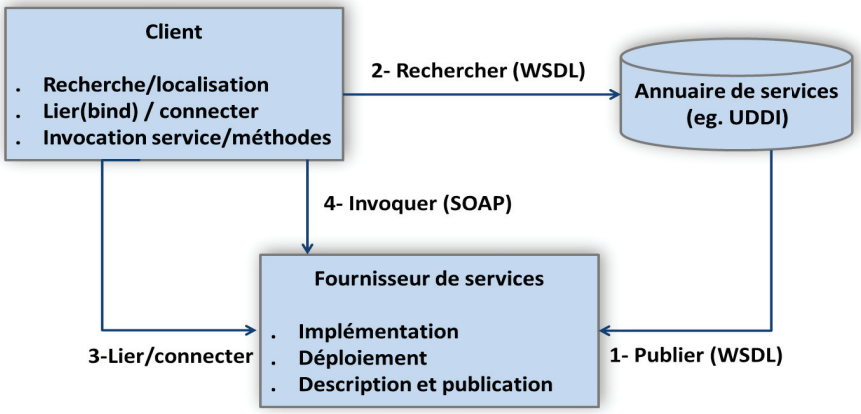


FIGURE 2.1 – **Interactions au sein d'une architecture orientée service**

2.1.2 Sûreté de fonctionnement (SdF)

Cette section a pour objectif d'introduire les concepts liés au domaine de la SdF permettant de cerner plus précisément la tolérance aux fautes. Selon Avizienis et al. [ALRL04] la sûreté de fonctionnement est l'aptitude ou propriété d'un système à délivrer un service de confiance justifiée. La notion de service délivré par un système correspond à son comportement perçu par son, ou ses utilisateurs (i.e., autre système en interaction avec celui considéré). Le service est dit *correct* si le service délivré accomplit la fonction du système décrite par la spécification fonctionnelle (i.e., ce à quoi le système est destiné).

2.1.2.1 Terminologie : défaillance, erreur, faute

Melliar-Smith [MSR77] est le premier à avoir fait la distinction entre la défaillance, l'erreur et la faute. Lorsque le service délivré dévie du service *correct*, un évènement, appelé *défaillance (ou panne)* survient. Il existe plusieurs raisons possibles liées à cette déviation : i) sa non-conformité à la spécification, ou ii) l'inadéquation de la spécification par rapport à la fonction du système. Une *erreur* est la partie de l'état du système susceptible d'entraîner une défaillance. La **propagation** d'erreur, appelée *délai de latence* se manifeste avant l'apparition d'une défaillance. La *faute* est la cause (ou **activation**) supposée ou adjugée d'une erreur et peut être le résultat (ou **conséquence**) d'une défaillance.

Pour des raisons d'abstraction, un système est souvent décomposé en sous-systèmes appelés composants, pouvant être à leur tour des systèmes. Pour un niveau d'abstraction donné, les composants sont considérés comme élémentaires. Une *défaillance* d'un de ces composants peut être source de *faute* pour le système composite, pouvant causer une *erreur* dans le système. La chaîne causale (*faute* $\longrightarrow$ *erreur* $\longrightarrow$ *défaillance*) devient alors récursive.

2.1.2.2 Classification des défaillances

Les défaillances des composants d'un système composite sont considérées comme les fautes à combattre. Avizienis et al. [ALRL04] caractérisent ces défaillances selon plusieurs points de vue (voir 2.2) :

- **Domaine.** Il permet de spécifier quatre types de défaillances selon le comportement du composant :
  - **Défaillance franche (Crash)**. Le composant cesse toute interaction avec les autres composants du système. La notion d'interaction dépend du modèle de système considéré (e.g., appel de procédure, envoi de message, écriture dans une mémoire partagée) ;
  - **Défaillance temporelle (Omission)**. Le composant interagit avec les autres composants du système en dehors des fenêtres temporelles attendues. Cela concerne aussi bien des interactions ayant lieu trop tard (i.e., échéance manquée) ou trop tôt. La défaillance par crash est une défaillance temporelle, dans le sens où toutes les interactions seront réalisées trop tard (ou jamais en l'occurrence).
  - **Défaillance en valeur**. Le composant interagit avec les autres composants du système avec des valeurs incorrectes.
  - **Défaillance Byzantine**. Le composant défaille de manière arbitraire. Ce mode de défaillance regroupe l'ensemble de tous les modes de défaillance ainsi que leurs combinaisons (e.g., valeur erronée trop tard).

- **Détectabilité.** Elle réfère à la propriété qu'une défaillance soit signalée ou non à l'utilisateur du système. Un mécanisme de détection permet par exemple de vérifier l'exactitude du résultat délivré. La signalisation d'une défaillance peut être communiquée à tord à l'utilisateur. La signalisation est alors dite *fausse alerte*.

- **Consistance.** Les défaillances sont dites *consistantes* si le service incorrect est perçu de façon identique par tous les utilisateurs du système. Sinon elles sont *inconsistantes*. Par exemple, les défaillances d'omission sont consistantes contrairement aux défaillances Byzantines.

- **Conséquence.** Les conséquences de défaillances sont classées selon leur niveau de gravité (ou sévérité) dépendant largement du type d'applications. En général, deux niveaux sont distingués (i.e., *mineur* et *catastrophique*) et se définissent par rapport au bénéfice fourni par le service rendu en absence de défaillances, et les conséquences de défaillances. Les défaillances sont dites *mineures* lorsque leurs conséquences ont un coût similaire à celui de la prestation du service correct. Les défaillances sont dites *catastrophiques* lorsque le coût de la défaillance est conséquente (e.g., perte humaine, crash financier).

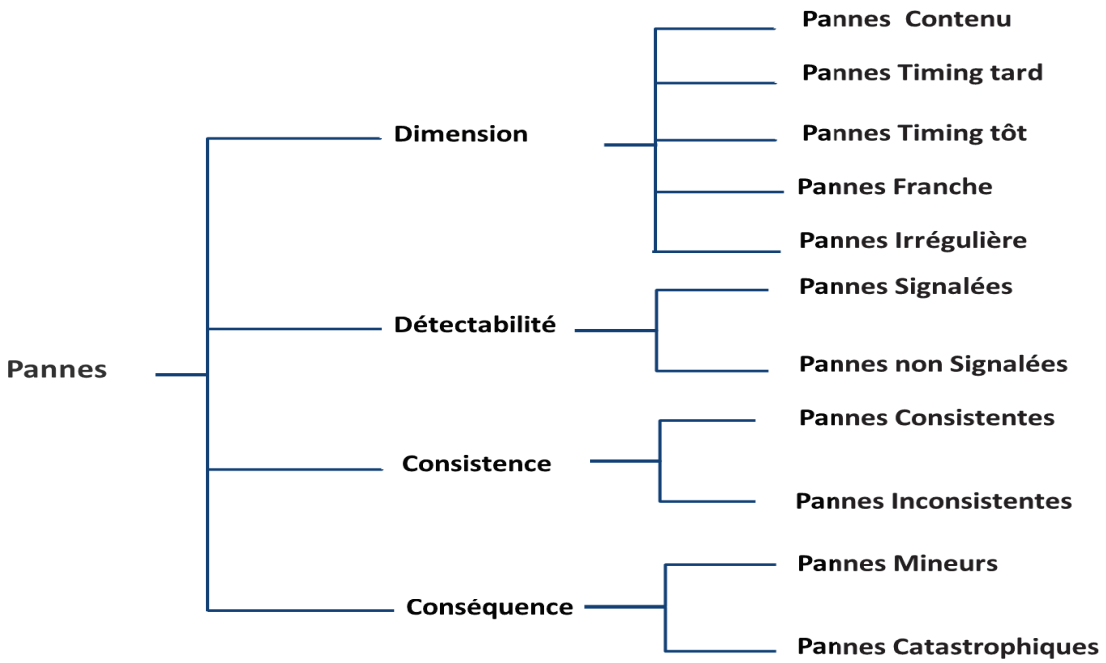


FIGURE 2.2 – **Classification des Défaillances dans les service Web**

2.1.2.3 Classification de fautes

Avizienis et al. [ALRL04] classifient les fautes élémentaires selon huit points de vue : *phase de création/occurrence, frontière du système, cause et persistance de la faute, dimension du système, objectif du concepteur/développeur ainsi que son intention et sa compétence* (voir 2.3).

Les fautes peuvent être soit introduites au moment de la conception et du développement du système (appelées *fautes de conception*) soit apparaître au moment de l'exécution du système (appelées *fautes opérationnelles*). Les fautes sont *internes* (resp. *externes*) si elles sont produites à l'intérieur (resp. extérieur) du système. La cause des fautes est *naturelle* si elles sont provoquées par un phénomène naturel. Elle est *non-naturelle* si l'être humain intervient dans le processus de création/occurrence de fautes. Les fautes persistent soit temporairement (dites *transitoires*) soit définitivement (dites *permanentes*). Par exemple, les composants défaillants re-fonctionnent normalement après la disparition des causes de la faute.

Les fautes peuvent être aussi définies par rapport à l'environnement de développement ou d'exécution (appelé dimension) du système. Ainsi, les fautes peuvent être soit de nature *logicielle* (e.g., composants défaillants) soit *matérielle* (e.g., machines défaillantes). Les fautes introduites par des humains sont *malicieuses* si l'objectif de ces derniers est de nuire au système tel que : accéder à des données confidentielles, ou dégrader le fonctionnement du système. Elles sont *non-malicieuses* dans le cas contraire. Les fautes sont *intentionnelles* si le concepteur/développeur a sciemment pris de mauvaises décisions mais sans aucun objectif de nuire le système. Les fautes sont accidentelles si le concepteur/développeur a pris des décisions dont les effets de bord sont involon-

taires. Les fautes peuvent être aussi dues à un manque de compétences de la part du concepteur/développeur.

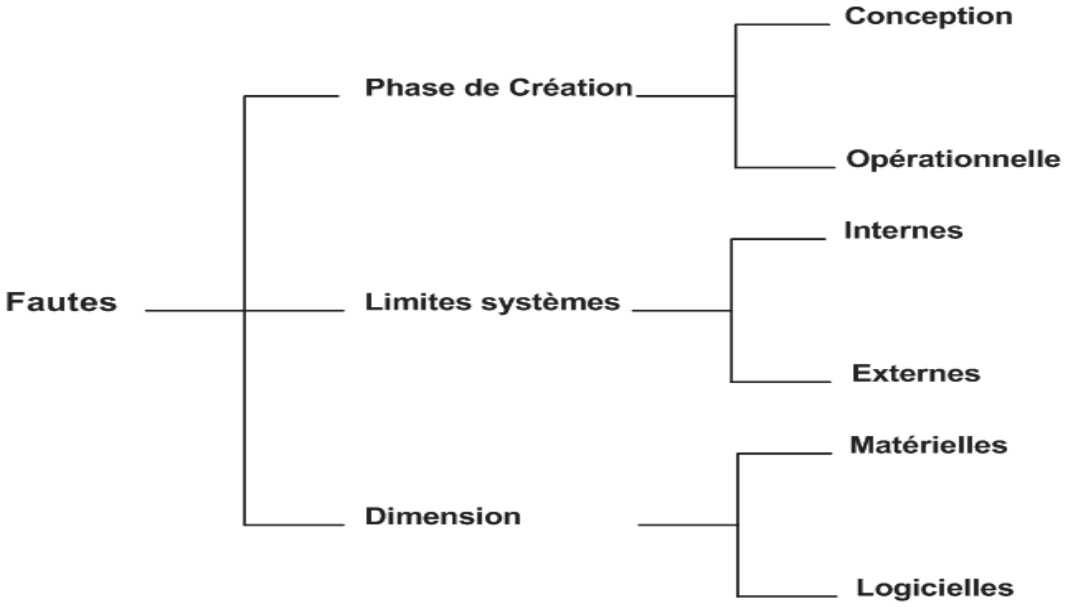


FIGURE 2.3 – **Classification des fautes**

2.1.2.4 Tolérance aux fautes en bref

La tolérance aux fautes est l'une des méthodes de SdF à laquelle nous nous intéressons dans cette thèse. Elle consiste à éviter la défaillance du système en utilisant des mécanismes *de détection et recouvrement/compensation d'erreur* causée par la défaillance d'un ou plusieurs composants (considérée comme des fautes pour le système global). Il est essentiel de bien cerner la nature des fautes que l'on cherche à tolérer afin de construire des mécanismes de tolérance qui contiennent la diversité appropriée pour être efficaces. De nombreuses classifications des fautes existent, chacune faisant appel à un traitement spécifique de tolérance aux fautes. Ainsi les fautes de conception ne peuvent être tolérées par une simple réplication du composant sur plusieurs machines, mais un développement de plusieurs versions de ce composant par des équipes différentes serait plus approprié.

La tolérance aux fautes vise à garantir la *fiabilité* de services et leur *disponibilité* dans un contexte réparti. Un service défaille s'il devient *non-fiable* et/ou *non-disponible*. La *fiabilité* est la capacité qu'un système distribué continue de fonctionner en dépit de fautes (i.e., présence de composants défaillants) [ALRL04]. Carter et al. [ACKL87] définissent trois facteurs dont dépend la fiabilité d'un système : i) *fiabilité des machines* sur lesquelles ses composants s'exécutent ; ii) *fiabilité* de ses composants ; et iii) *fiabilité* des liaisons réseau et du degré de congestion et de collision. La fiabilité se mesure comme la probabilité qu'un système ne défaille sur une periode d'exécution $[t_1, t_2]$ (voir Equation 2.1) où $MTTF$ est le temps moyen jusqu'à la prochaine défaillance (*en anglais*, Mean Time

To Failure). $MTTF$ correspond à une densité de probabilité de défaillances, avec une distribution exponentielle, pour décrire l'occurrence des défaillances dans le système.

$$F(t_1 < X \leq t_2) = 1 - \int_{t_1}^{t_2} MTTF(t)dt \tag{2.1}$$

La *disponibilité* se réfère, quant à elle à la capacité d'un système à fournir des résultats au moment où ils sont nécessaires ou utiles à l'utilisateur [ALRL04]. Pendant la période d'exécution, le système est dans un état soit *actif* soit *en réparation*. Réparer un système consiste à le remettre en état pour rendre un service correct). La disponibilité se mesure comme la probabilité qu'un système soit disponible (i.e., fournisse un service correct) sur une période $[t_1, t_2]$ (Equation 2.2 où $MTTR$ est le temps moyen de réparation (*en anglais*, Mean Time To Repair) et $MTBF$ est le temps moyen entre deux défaillances ($MTTF + MTTR$) (*en anglais*, Mean Time Between Failures).

$$A(t_1 < X \leq t_2) = \int_{t_1}^{t_2} \frac{MTTF(t)}{MTBF(t)}dt \tag{2.2}$$

En dépit de cette grande hétérogénéité de situations de fautes, différents aspects de tolérance aux fautes ont été abordés par deux grandes familles de techniques : *préventive* (i.e., éviter les erreurs) et *curative* (i.e., corriger les erreurs). La *redondance* et la *diversité* sont deux techniques préventives incontournables dans le domaine de la tolérance aux fautes. La technique de *redondance* consiste à construire dans un système une sorte de "*capacité additionnelle*" de service, *partielle* (e.g., correction d'erreur ou vérification en ligne d'une condition de validité) , ou *totale* (e.g., déploiement sur plusieurs machines). La redondance est utilisée aussi bien pour la détection d'erreur que pour leur compensation. En effet, des erreurs sont détectables par exemple, lors de la comparaison de résultats fournis par plusieurs répliques. Aussi, l'état erroné d'un système possède une redondance suffisante pour masquer l'erreur, permettant ainsi la poursuite de son exécution. Par exemple, une des répliques reprend le fonctionnement du composant défaillant à partir du dernier *checkpoint*.

La technique de *diversité* offre quant à elle une "*capacité additionnelle*" de service indépendante du service à dupliquer d'un point de vue de la création et de l'activation des fautes [Lap04]. En effet, aucune redondance ne peut protéger le composant en question de toute type de faute conduisant à des défaillances en mode commun (i.e., de manière identique et au même moment). À partir de ce constat, le concept de *diversité* devient essentiel, et est étroitement lié aux fautes à tolérer. Contrairement à la redondance pour laquelle une formalisation des protocoles est bien établie, la diversité reste un terrain fertile de recherche.

La seconde famille fait quant à elle appel à un mécanisme de *diagnostic des fautes* pour identifier le composant du système responsable de l'erreur ayant conduit à la défaillance d'un autre composant. Une fois le diagnostic établi, un mécanisme de recouvrement se charge de reconfigurer le système ou de ré-exécuter les composants responsables

de l'erreur et ceux défaillants, de les retirer ou les remplacer. Aussi, deux formes de recouvrement d'erreur existent :

- *Reprise* : le système revient à un état antérieur cohérent à partir duquel il reprend son fonctionnement.
- *Poursuite* : le système continue son fonctionnement souvent de manière dégradée, à partir d'un état ultérieur. Ce mode de recouvrement est très dépendant du type d'application.

Nous venons de présenter les grands principes de la tolérance aux fautes. Nous allons maintenant nous intéresser à l'utilisation de ces principes dans le cas particulier des services Web.

2.2 Tolérance aux fautes dans un environnement SOA

Le développement des applications basées service tolérantes aux fautes font souvent appel à des solutions capables de faire face à des crashs et/ou comportements anormaux des services Web. Par la suite, nous introduisons une taxonomie des fautes spécifiques à SOA ainsi que les différents types de détection d'erreur et de recouvrement associés.

2.2.1 Taxonomie des fautes dans SOA

Plusieurs taxonomies de fautes, raffinant celle de Avizienis et al. [ALRL04] ont été présentées pour des applications logicielles distribuées classiques (e.g., [DGR04]). Le principal intérêt de ces taxonomies est de construire des modèles de fautes (certains dépendants du domaine d'application) nécessaires à la *détection d'erreur* et au *diagnostic de fautes*. L'identification des classes de fautes permet d'identifier des patterns de réaction (e.g., recouvrement) au lieu de devoir prendre des décisions au cas par cas.

Cependant, les caractéristiques des services Web diffèrent considérablement de celles des applications classiques. Les services Web sont faiblement couplés et dynamiques de nature, et interagissent de manière dynamique sans une connaissance a priori des uns des autres. Par conséquent les fautes (i.e., défaillances au niveau du composant) dans un environnement SOA peuvent être fréquentes. Ainsi, une étape clé pour un mécanisme de détection/diagnostic est d'identifier les différents types de fautes dans un tel environnement et de fournir une taxonomie claire de tous ces types.

Chan et al. [CBS+09] proposent une taxonomie de fautes dans une composition de services Web. La taxonomie proposée peut être utilisée pour différencier les différentes défaillances observées, et par conséquent constitue une aide pour le recouvrement. Les auteurs subdivisent les fautes en une autre dimension, en plus de celle de Avizienis et al. [ALRL04], avec comme catégories : *physiques*, *de développement* et *d'interaction*. La première catégorie comporte les défaillances au sein de l'infrastructure du réseau ou côté

serveur. Dans la seconde, les fautes sont introduite dans le système par son environnement (e.g., paramètre incorrect, changement d'interface). Finalement, nous retrouvons les fautes de contenu (i.e., les résultats délivrés par le service composite sont différents de ceux attendus) et de timing (i.e., les résultats sont fournis en dehors de l'intervalle prévu de temps). Les auteurs combinent les deux dimensions pour leur associer des effets observables (i.e., défaillances), correspondant aux causes des fautes.

Dans la littérature Bruing et al. [BWM07] proposent une taxonomie de fautes typiques dans SOA. Les fautes étudiées incluent les fautes *matérielles*, les fautes *logicielles*, les fautes *réseaux*, et les fautes d'*interactions*. Les auteurs partent du fait que les fautes peuvent apparaître au cours des différentes étapes SOA : `publication`, `découverte`, `composition`, `binding`, et `exécution`. Si ces fautes sont actives, elles causeront des erreurs conduisant à une défaillance à moins que la structure du système soit capable de traiter ces erreurs (e.g., masquage de fautes via une redondance/diversité de service). La taxonomie commence par les classes de fautes les plus générales, associées chacune à une étape SOA, jusqu'à celles plus spécifiques, offrant ainsi une plus large couverture de toutes les fautes possibles. Cette taxonomie peut être raffinée pour chaque domaine d'application en des fautes plus spécifiques. Elle permet de fournir une connaissance sur la manière à déployer des services Web fiables. Cependant, une implémentation de cette taxonomie pour détecter les erreurs dans le SOA reste une question ouverte.

Cheun et al. [CLK12] définissent une extension de cette taxonomie en considérant les caractéristiques du paradigme service (e.g., entités faiblement couplées [PTDL03]), les standards SOA (e.g., SOAP [SOA02], WS-BPEL [Org07]), et les éléments SOA (e.g., Enterprise Service Bus [Cha04], moteur BPEL [Jur06]). Ils identifient trois types de fautes : i) *fautes imbriquées* (e.g., signature d'opération avec des types de données non-définis) ; ii) *fautes d'interaction* (e.g., inconsistence et/ou incompatibilité dans le binding entre deux composants) ; et iii) *fautes exécutives* (e.g., paramètre ou ressource requise au moment de l'exécution). Les auteurs proposent une formalisation de cette taxonomie ainsi qu'un prototype pour gérer les fautes de service.

Dans cette thèse, nous nous sommes intéressés plus particulièrement aux fautes introduites aux étapes suivantes : i) `exécution` (e.g., entrée incorrecte (*incorrect input*), faute dans le service *service faulty*, faute serveur/réseau) ; ii) `publication` (e.g., description incorrecte du service (*service description fault*) donnée par le fournisseur de manière malicieuse (e.g., niveau de réputation biaisé) ou pas (e.g., écriture ambigue d'une date (DDMMYY ou YYMMDD))) ; et iii) `binding` (e.g., liaison à un mauvais service (`Bound to wrong Service`). Ces fautes peuvent conduire Ãă des défaillances soit franches ou byzantines.

2.2.2 Mécanismes de tolérance aux fautes dans SOA

Dans cette section, nous nous intéressons aux mécanismes nécessaires à la mise en œuvre de stratégies de tolérance aux fautes : *détection d'erreur* et *recouvrement/compensation*

d'erreur.

2.2.2.1 Détection d'erreur

La détection d'erreur dans les services Web fait l'objet de plusieurs travaux de recherche (e.g., [YD07, LHS08, BLGC10, FCMJ12]).

Dans [YD07], Yan et al. suggèrent de réutiliser la masse de connaissance sur la détection d'erreur dans les systèmes industriels à évènements discrets. Ils présentent une discussion sur la détection d'erreur dans les orchestrations de services Web au moment de l'exécution. Les auteurs supposent que les défaillances ne sont pas observables. La granularité dans le modèle d'exécution est au niveau processus (i.e., services Web). A base de cette discussion, Li et al. ont proposé une nouvelle approche de détection d'erreur [LHS08]. L'idée principale est de considérer les processus WS-BPEL comme des systèmes à évènements discrets et les faire correspondre à des réseaux de Petri colorés. Les erreurs considérées proviennent des services Web défaillants et des données potentiellement corrompues dans des orchestrations de processus WS-BPEL.

Dans [BLGC10], Borrego et al. proposent un framework de détection d'erreur pour des processus métier décrits avec *BPMN*, contenant une couche de diagnostic. Les auteurs considèrent des processus composés de plusieurs activités susceptibles de renvoyer des résultats incorrects, dus à des données en entrée incorrectes. La fonctionnalité correcte de chaque activité est spécifiée par un sensemble de règles de conformance. Ces règles sont transformées en des problèmes de satisfaction de contraintes. Les erreurs sont alors détectées au moyen d'un résolveur de contraintes.

Dans [FCMJ12], Frantz et al. proposent une solution pour détecter les erreurs dans les solutions EAI (acronyme de *Enterprise Application Integration*). Ces solutions sont des types de workflows dans lesquels les messages provenant d'un sous-ensemble d'applications sont acheminés vers des processus pour éventuellement les transformer et transmettre les résultats à un autre sous-ensemble d'applications. Ces solutions EAI peuvent être spécifiées comme une orchestration ou bien une chorégraphie. Dans une orchestration, un processus central a une vue globale et précise de l'état d'exécution de la solution EAI. Un *Gestionnaire d'évènement* capture les succès/échecs de lecture/écriture des messages sous forme d'évènements. L'échange des messages entre les processus et les relations parent-fils sont stockés dans un graphe. Un *Détecteur d'erreur* analyse ce graphe pour trouver les corrélations dans lesquelles un message spécifique est impliqué et les vérifier (e.g.,erreurs de communication ou de deadline, erreurs structurelles). Un exemple d'erreurs structurelles sont les corrélations pour lesquelles des messages sont manquants ou sont plus que prévu). Contrairement aux autres approches, cette solution est indépendante du modèle d'exécution (processus ou tâche) et de la forme de la composition (orchestration ou chorégraphie).

2.2.2.2 Recouvrement et compensation d'erreur

Depuis une décennie, les mécanismes pour rendre les compositions de services Web tolérantes aux fautes ont suscité un vif intérêt dans la communauté de recherche. Les approches existantes se sont principalement intéressées au recouvrement par reprise, et plus spécifiquement aux transactions. Cependant, l'autonomie des services Web et la latence du Web ont influencé considérablement l'orientation de recherche exigeant des modèles transactionnels et techniques de recouvrement par reprise plus flexibles et complexes.

Recouvrement par reprise. Le modèle transactionnel classique a prouvé son succès dans l'application de la SdF aux systèmes distribués fermés. Il est beaucoup exploité pour implémenter des services Web primitifs (i.e., non-composites). Cependant, il s'avère inapproprié pour rendre les compositions de services Web tolérantes aux fautes pour deux raisons : **(i)** La gestion des transactions (réparties sur l'ensemble des services Web) demande une coopération parmi les supports transactionnels des services Web individuels. **(ii)** Le verrouillage des ressources jusqu'à la terminaison de la transaction imbriquée est en général inappropriée pour les services Web (clients avec des délais d'attente très restreints). Une solution pour ce problème est les modèles transactionnels améliorés, faisant référence à des transactions imbriquées ouvertes (décomposition de la transaction en sous-transactions pouvant s'engager indépendamment. En cas d'échec de la transaction, la vérification de la propriété de l'atomicité ($ACID$), demande une compensation des sous-transactions engagées dans cette transaction. Cependant, les services Web doivent fournir des opérations de compensation pour toutes les opérations qu'ils fournissent (langages BPEL et WSCI). Il est clair qu'une cascade de compensation est à prévoir lors d'une cascade d'annulation (e.g., [LCR06]). En plus des solutions côté-client à la coordination des transactions distribuées imbriquées, des auteurs comme Hass et al. [HB04] (WS-Transaction) proposent des protocoles transactionnels distribués supportant le déploiement des transactions sur le Web sans imposer des verrous trop longs sur les ressources Web.

Recouvrement par poursuite. Le mécanisme de gestion d'exception est largement utilisé pour la mise en œuvre de ce type de recouvrement dans les compositions de services Web. Dans BPEL, les activités peuvent être dotées de Handlers d'Exception qui assure la continuation de l'exécution en cas d'erreur. Cependant, quand une activité au sein d'un processus concurrent signale une exception, toutes les autre activités intégrées terminent, et seulement le handler correspond est exécuté. Par conséquent, le recouvrement d'erreur prend en compte réellement qu'une seule exception et n'est pas en mesure d'assurer un recouvrement à un état correct. Le seul cas possible est lorsque l'effet de toutes les activités en échec est défait (Pas possible dans un contexte de services Web). Une solution à ce problème est de structurer la composition de services Web en termes d'actions atomiques coordon-

nées (en anglais, Coordinated Atomic (CA) Actions). Dans [RRS+97], les actions atomiques sont utilisées pour contrôler la concurrence coopérative et implémenter un recouvrement coordonné d'erreur sont utilisées pour contrôler la concurrence coopérative et implémenter un recouvrement coordonné d'erreur. Deux cas de figure se présentent : i) les participants atteignent la fin de l'action et produisent un résultat normal ; ii) en présence d'exception(s), ils sont tous impliqués dans leur handling coordonnée. Si la gestion d'exception réussit, l'action finit avec succès. Mais si le handling n'est pas possible, alors la responsabilité pour le recouvrement est transmise à l'action contenante où une exception d'action externe est propagée. Les CA actions fournissent un mécanisme pour développer des services Web composite tolérants aux fautes : une CA action spécifie la réalisation collaborative d'une fonction donnée par des services composés, et les services Web correspondent à des ressources externes. Cependant, comme pour les transactions, les propriétés ACID sur des ressources externes ne sont pas appropriées dans le cas de services Web. Tartanoglu et al. [TIRL03] introduisent la notion de Web Service Composition Action (WSCA) qui relaxe les besoins transactionnels sur les ressources externes.

Recouvrement par réplication La réplication de services Web services avec état peut être réalisée de plusieurs manières : (i) active (ou machine à états) [LCR06] ; (ii) passive (ou copie primaire) [MSR77] ; iii) semi-active (i.e., combinaison des deux). La réplication active est basée sur l'idée d'envoyer la requête à tous les réplicas et attendre toutes les réponses. Ce type nécessite l'ordonnancement de la requête, et la suppression des invocations imbriquées. Elle garantit une synchronisation des états, cependant, elle suppose que toutes les opérations produisent des résultats et transitions d'état déterministes. Cette limitation peut être contournée en déclarant explicitement toutes les fonctions non-déterministes et redirigeant les invocations vers un unique service. Ce dernier est responsable de la diffusion des changements de l'état interne à tous les autres réplicas. Cette approche, appelée semi-active est une composition des idées des deux approches machine à états et copie primaire. Dans la technique copie primaire, toutes les requêtes sont envoyées à un seul service appelé primaire. Ce service met à jour automatiquement tous les réplicas secondaires. Quand il défaille, une réplique secondaire est sélectionnée comme nouveau primaire. En vertu de la possibilité de réseaux ad-hoc de se diviser et fusionner, il devient évident que les approches actives et semi-active sont inappropriées (synchronisation très coûteuse pour maintenir tous les réplicas dans le même état interne, besoin de maintenir le trafic du réseau aussi bas que possible). Dans [DJ07] les auteurs utilise la technique copie primaire pour réaliser leur système de réplication. La synchronisation des répliques s'effectue via un protocole (Simple Replicator Protocol (SRP)). Un réplicateur de services Web est conçu comme une collection de modules coopérant via une base de données pour stocker toutes les données pertinentes, pour structurer la tâche de réplica-

tion. Les modules consomment peu de mémoire et font partie de toute instance du réplicateur. Le résultat est une communauté pair-à-pair de nœuds dans laquelle les élections décide de la distribution des tâches nécessaires, de préférence sur les nœuds les plus puissants, tandis que d'autres restent oisifs.

Dans cette thèse, nous nous intéressons à la compensation par diversité faisant l'objet de recherches actives dont nous analysons un échantillon de travaux en Section 2.3.

Compensation par diversité. Traditionnellement, la recherche en tolérance aux fautes logicielle s'articule autour de deux approches : N-versioning (en anglais, N-Version programming (NVP)) de Avizienis [Avi85] et blocs de recouvrement (en anglais, Recovery Blocks (RB)) de Randell [Ran75]. Dans la première approche, N-versions d'un programme exécutent de multiples implémentations d'une même fonctionnalité mais ayant des conceptions diverses en parallèle (i.e., différentes équipes de développement, différents outils). Un vote majoritaire est effectué pour obtenir les résultats final à retourner à l'utilisateur (ou application appelante). Dans la seconde approche, les RB's invoquent des implémentations redondantes de manière séquentielle si la sortie d'un RB ne réussit pas le test de conformité/validité. La technique de NVP semble être une bonne candidate aussi bien à des objectifs de SdF que de performance, malgré le coût de développement de version multiples. Dans le domaine des services Web, Looker et Munro [LMX05] ont été les premiers à introduit NVP. Sommerville [SHD05] a suivi avec son approche de tolérance aux fautes à base de conteneur configurable avec une politique spécifiant le type de mécanisme de tolérance aux fautes appliqué aux services le contenant. Dobson [DHS05]a utilisé quant à lui les mécanismes fournis par WS-BPEL pour supporter NVP et RB.

2.3 Travaux existants sur la tolérance aux fautes

Dans cette section, nous allons nous focaliser sur les études s'intéressant à différents aspects problématiques de la technique de compensation sous ses deux formes : réplication et diversité de services Web. Durant la dernière décade, un grand nombre de travaux ont mis l'accent sur la première forme, laissant ainsi un champ fertile de recherche à explorer pour l'application du concept de diversité à la conception de services Web tolérants aux fautes. Rappelons que l'un des avantages de la diversité est d'éviter que les services Web sémantiquement équivalents (i.e., versions de service conçues/développées indépendamment par différents fournisseurs) ne défaillent pas de la même manière et au même moment contrairement à une réplication classique (i.e., minimisation considérable de la probabilité d'occurrence de fautes communes parmi les différentes versions).

2.3.1 Approches basées sur la réplication

2.3.1.1 ReplicationManager

Chan et al. [CLM06] présentent une gestion dynamique de la réplication afin d'améliorer la sûreté de fonctionnement des services Web. Fig. 2.4 montre l'architecture proposée pour tolérer les défaillances *franches*.

Elle se base sur la technique de réplication des Services Web en mode passif. Le composant majeur de cette architecture est le Gestionnaire de réplication, agissant comme le coordinateur des services Web. Il est responsable de la création des services Web, du choix du service Web primaire (plus rapide, plus robuste..etc) utilisant un algorithme de sélection, de l'enregistrement du *WSDL* et *UDDI*, de la vérification de la disponibilité du service Web primaire et enfin de la sélection de nouveau primaire si le service primaire échoue.

Les services Web sont répliqués et déployés sur différentes machines, mais seul un service Web, appelé primaire fournit la réponse à une requête. Le gestionnaire de réplication (en anglais, ReplicationManager (`GR`)), qui représente la partie centrale de cette architecture, sélectionne le premier service primaire à invoquer. La vérification de la disponibilité de ce service est effectué par la technique du chien de garde (en anglais, *Watchdog*). Cette technique consiste à faire plusieurs tentatives de retransmission de la requête avant de déclarer la panne du service.

Si la panne du service primaire est avérée, le GR utilise un algorithme appelé *Anycasting algorithme* pour sélectionner un nouveau primaire. `GR` se charge alors de mettre à jour le `WSDL` par l'adresse du nouveau service Web primaire. Ainsi, les clients peuvent toujours accéder au service Web avec la même `URL` et ce de manière transparente en dépit des défaillances. Cette approche est dynamique dans la mesure où une réplique s'engage à la fois pour être exécutée, tandis que les autres répliques se trouvent dans un état en veille. Si le service primaire échoue, une autre réplique peut être utilisée immédiatement avec peu d'impact sur le temps de réponse.

Elle présente plusieurs avantages à savoir le basculement transparent vers un autre service dans le cas d'échec de premier, des expérimentations qui montrent comment la redondance améliore la disponibilité des services Web. Cependant, cette approche n'est pas applicable pour le cas des fautes Byzantines. Pour cela, elle devrait posséder une spécification sur la valeur de résultat à retourner.

2.3.1.2 ServiceGlobe

Keidl et al. [KSK03] proposent une architecture flexible, nommé `ServiceGlobe` pour supporter une exécution fiable des services pouvant être intégrée (ou plug-in) dans les plates-formes existantes de services. Le `dispatcher` représente le cœur de cette architecture, il fourni des services de réplication automatique. Ces services peuvent être interne ou externe, les services internes sont spécifiques et natifs pour l'architecture

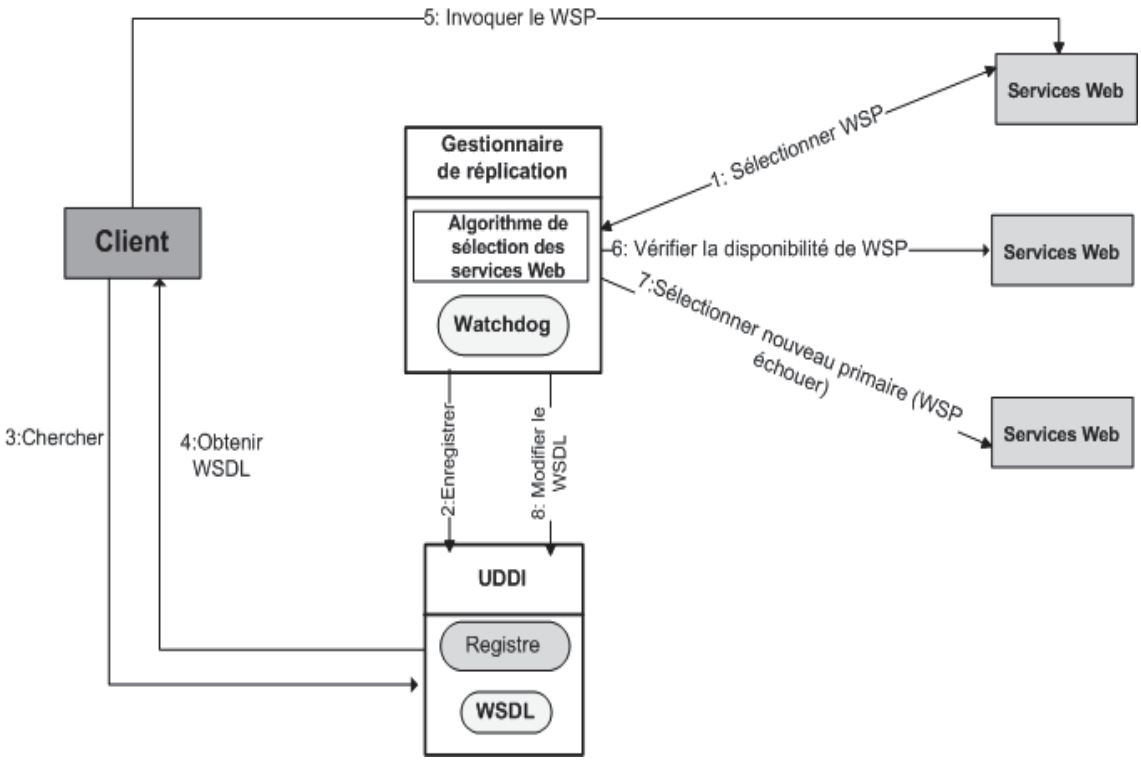


FIGURE 2.4 – **Architecture proposée pour la sûreté de fonctionnement dans les services Web**

`ServiceGlobe`, et les services externes sont les services actuellement déployés sur Internet, qui ne sont pas fournis par `ServiceGlobe` lui-même. Une réplication automatique schématisée dans la Fig. 2.5 permet d'assurer une haute-disponibilité des services Web. La réplication automatique consiste à basculer d'une manière automatique les instances exécutées sur des machines avec une grande charge du travail vers des machines moins chargées. Cette technique a des avantages : Premièrement, le développement des services sans avoir à considérer la haute disponibilité et l'équilibrage de charge est beaucoup plus facile. Deuxièmement, chaque service existant peut être transformé facilement en un service disponible, comme il n'y a pas de partage de données entre les différentes instances de ce service ou il a une sorte de contrôle de concurrence, par exemple l'utilisation d'une base de données comme un back-end.

En effet, `ServiceGlobe` permet de faire basculer des instances d'exécution de services Web sur des machines avec une grande charge de travail vers d'autres moins chargées afin de respecter un certain équilibre de charge. Ce basculement est effectué de manière transparente.

Les auteurs présentent aussi une technique de sélection dynamique des services en se basant sur une description formelle d'interfaces, appelée `TModel`. La description de chaque service est affecté à un `TModel` dans l'UDDI. Par conséquent, un service peut être invoqué comme une instance du `TModel` associé. Dans la sélection dynamique des services, une application n'a plus besoin de connaitre les points d'accès codés en

dur pour invoquer un service. Il suffit de faire référence au `TModel` associé. Ainsi, l'application doit connaitre la fonctionnalité à invoquer au lieu du service mis en œuvre pour réaliser cette fonctionnalité. Le `TModel` fournit une couche d'abstraction du service Web réel, ainsi que les contraintes permettant à un service Web d'influencer la sélection dynamique [KSK].

Les différents modes d'appel à un `TModel` correspondent aux deux modes classiques d'invocation : i) *passif* : la variante, appelée `Unicast` active un seul service ; et ii) *actif* : deux variantes sont proposées : `Multicast` où un sous-ensemble de services est activé, et `Broadcast` où tous les services sont invoqués.

Pour mettre en œuvre la technique de réplication automatique et celle de sélection dynamique, les auteurs développent un composant, appelé `dispatcher` agissant comme un *proxy* pour les services. Ce `dispatcher` se charge de surveiller un ensemble d'hôtes sur lesquelles les mêmes instances des services s'exécutent pour équilibrer la charge de travail sur chaque hôte. Si aucun hôte n'est disponible, le `dispatcher` peut enregistrer les messages entrants dans un *buffer*, ou bien les rejeter en envoyant une réponse "**temporairement indisponible**" au demandeur du service. Ce choix est fait en fonction de la configuration du `dispatcher`.

Cette approche basée sur l'invocation des instances identiques de service améliore la disponibilité des services Web. Cependant, elle se limite à des fautes liées au serveur et non pas au service lui-même.

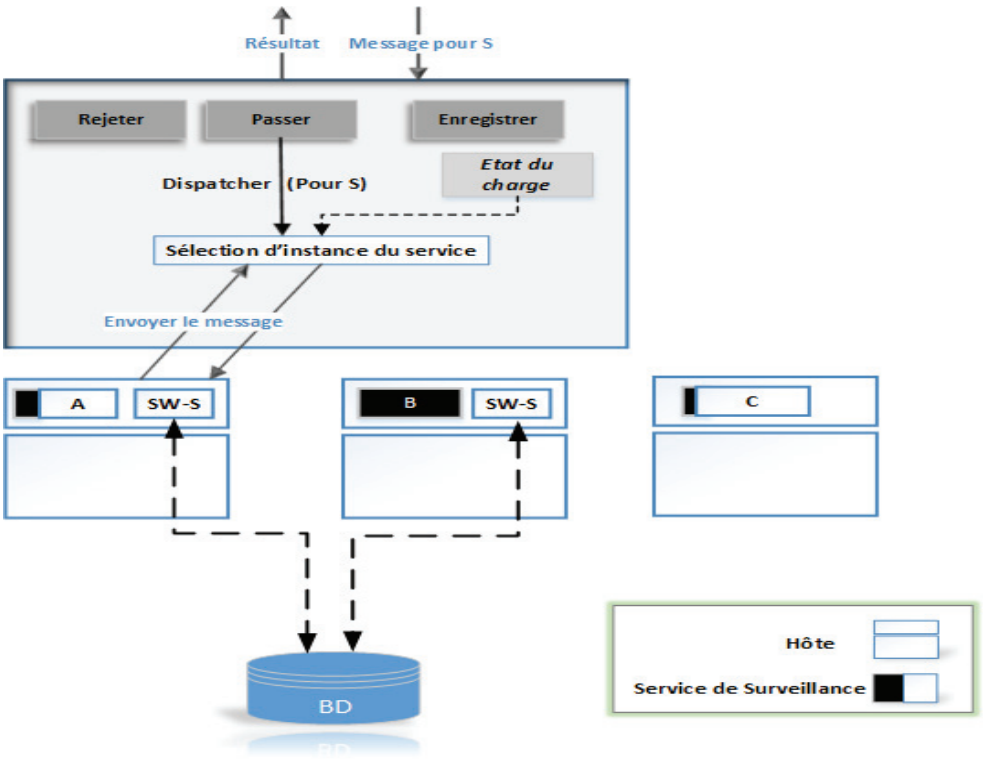


FIGURE 2.5 – **Architecture de dispatcher**

2.3.2 Approches basées sur la diversité

2.3.2.1 Connecteurs

Salatge et Fabre [SF07] proposent une infrastructure nommée IWSD (acronyme de "*Infrastructure for Web Services Dependability*") pour tolérer les défaillances *franches* et *en valeur* de services Web. Elle se base sur des connecteurs SFTC (acronyme de "*Specific Fault Tolerance Connectors*") assurant une communication fiable entre les clients et les prestataires de service (voir Fig. 2.6). Cette architecture est composée des éléments suivants : Le connecteur qui est l'élément central de la plate-forme. Il traite les requêtes et les réponses fournies par les clients et les prestataires contactés en effectuant les actions de tolérance aux fautes définies par l'utilisateur. Le support d'exécution qui peut être considéré comme une machine virtuelle sur laquelle s'exécutent les connecteurs. Il est composé de deux modules distincts : (i) Le serveur d'exécution qui intercepte les requêtes des clients et charge le connecteur souhaité à partir de l'analyse du message reçu, et (ii) Le moniteur de surveillance qui est en charge de vérifier l'état de la plate-forme IWSD et d'évaluer l'état courant des Services Web en exécution. Le dernier composant est Le serveur de gestion permet de créer et gérer les comptes utilisateurs. Il permet également la création et le stockage des connecteurs associés à chacun d'eux. IWSD contrôle et exécute les connecteurs dans des applications critiques. cette tâche du contrôle est associé au serveur d'exécution qui fournit trois services : **le Service d'exécution** qui est le responsable de la vérification de la syntaxe des messages (Une exception SOAP est automatiquement générée dans le cas d'une réception d'un message mal-formé), de l'authentification et de la récupération de toutes les informations relatives au connecteur désiré. Le deuxième composant, **Gestionnaire des connecteurs actifs** est le mécanisme de stockage interne des connecteurs propres à un serveur d'exécution. Il contient les connecteurs déjà utilisés. Ces connecteurs sont téléchargés à partir du serveur de gestion. Le troisième composant est **le Chien de garde** qui a pour seule fonction d'envoyer des messages de type "I'm alive" au moniteur de surveillance et à son serveur de secours. Ce module permet de détecter si le serveur d'exécution est en arrêt ou non.

Ces connecteurs implémentent plusieurs variantes des stratégies passive et active, utilisant aussi bien des répliques identiques que sémantiquement équivalentes.

Pour la spécification et la mise en œuvre des SFTC, les auteurs introduisent un langage spécifique appelé DeWel. Ce langage offre les abstractions et les notations appropriées capables de réaliser des actions de tolérance aux fautes efficaces par des mécanismes de recouvrement ou de signalement d'erreur. Des assertions en termes d'exceptions SOAP peuvent être déclarées dans le connecteur. Lorsque ces dernières ne sont pas satisfaites, des mécanismes de gestion des erreurs (e.g., collecter des informations sur les erreurs et politiques de recouvrement) peuvent être activées pour traiter les exceptions au sein du service et celles se produisant lors de la communication entre clients et prestataires de service.

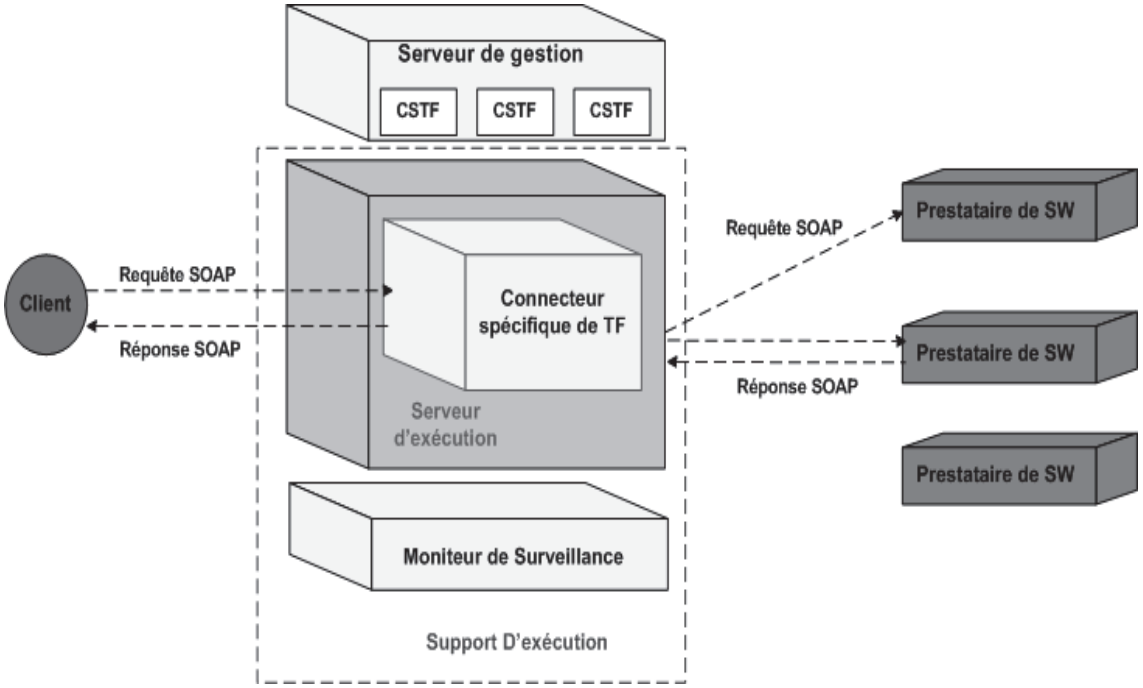


FIGURE 2.6 – **IWSD, une infrastructure pour la sûreté de fonctionnement des Services Web**

2.3.2.2 Containers

Dobson et al. [DHS05] proposent une approche basée sur les `Containers` de tolérance aux fautes dans les architectures `SOA` (voir Fig. 2.7). Les auteurs considèrent plusieurs exemples de fautes tels que l'insuffisance de ressources, des fautes dans la mise en œuvre des services et la perturbation du réseau. Ils utilisent la technique de diversité des services composant les applications `SOA`. Un `Container` joue le rôle d'un intermédiaire entre le client et les fournisseurs des services Web.

Le `Container` agit comme un *proxy* pour les services réels. Par conséquent, un message en route vers un service déployé sera intercepté par le `Container`, ce qui ajoute la caractéristique de la tolérance aux fautes au service. Cette action est transparente par rapport au client et fournisseurs du service [HSS+04]. Le `Container` est configuré avec une politique précisant le type de mécanisme de tolérance aux fautes à appliquer aux services le contenant (ré-exécution du service (retry), Recovery-Bolck [Ran75], redondance).

Le fonctionnement d'un `Container` ressemble à celui d'un serveur d'application *EJB*, il déploie des objets *proxy*, sous la forme d'un service transparent. Ces services de *proxy* interceptent les messages envoyés à des services réels. Cette interception est réalisée à travers le déplacement du *endpoint* (il représente l'adresse de service habituellement représenté sous la forme d'un URI), le *endpoint* du service réel est remplacé par le *endpoint* du service de *proxy*. Le client doit connaitre le *endpoint* du service de *proxy*. Contrairement aux composants *EJB*, les services "contenus" ne doivent pas résider à

l'intérieur du `Container`.

La diversité des services est fournie en mettant en liaison les services d'un `Container` et le marché des services. De cette façon, le recouvrement peut être réalisé à un coût faible. Toutefois, la solution proposée n'est pas encore assez mûre pour une mise en application réelle. Par exemple, la notion de similarité entre les services Web est très succinctement abordée. En outre, les mécanismes de recouvrement proposés ne sont pas appropriés aux services avec état.

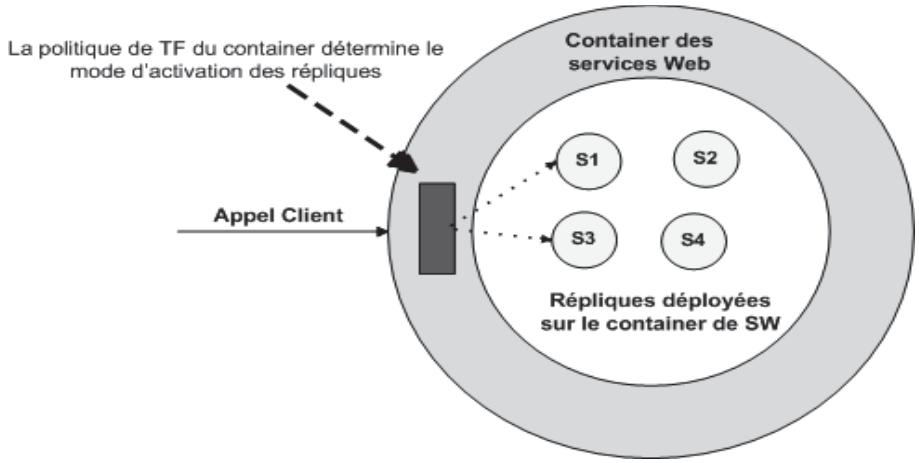


FIGURE 2.7 – **Le principe du container des Services Web**

2.3.2.3 Composition horizontale et verticale

Gorbenko et al. [GKR09] présentent une analyse de modèles de compositions fiables de services Web selon deux perspectives : *verticale* en termes de fonctionnalités et *horizontale* en termes de sûreté de fonctionnement (voir Fig. 2.8). Selon la perspective *verticale*, les auteurs utilisent la réplication et la diversité des services Web reliés par des patterns de composition. Les différents patterns sont : (i) `Reliable concurrent execution`; Toutes les répliques (versions) sont invoquées concurremment, et leurs réponses sont adjugées par le voteur. (ii) `Fast concurrent execution`; Tous les services disponibles sont invoquées simultanément, mais le résultat du premier service correcte est retourné, cette approche améliore le temps de réponse (iii) `Adaptive concurrent execution`; Tous (ou certaines) répliques (ou versions) sont exécutées simultanément. Le middleware est configuré pour attendre un certain nombre de réponses, mais pas plus d'un délai prédéfini.(vi)`Sequential execution`; la réplique (version) d'un service Web n'est invoquée sauf si la réponse reçue par le service primaire est incorrecte. Selon la perspective horizontale, l'objectif est d'améliorer la tolérance aux fautes particulièrement la disponibilité des services Web. Cette amélioration est réalisée grâce aux différents patterns utiliés, ainsi que l'utilisation des services Web similaires. L'architecture avec la composition horizontale comprend un composant appelé médiateur pour le vote parmi les réponses de tous les services Web divers et renvoie une réponse adju-

gée au client. Les auteurs définissent plusieurs objectifs de SdF à atteindre telles que la disponibilité, l'exactitude et la qualité de la réponse.

Ils proposent des modèles sous forme de protocoles de SdF pour des compositions fiables, ainsi que différentes stratégies pour invoquer des services identiques ou similaires en mode séquentiel ou parallèle, avec des ébauches de procédure non formelles pour le vote des réponses. Cependant, des patterns supplémentaires de composition doivent être mis au point pour implémenter ces protocoles. Aussi, de nouvelles activités de contrôle permettant au processus métier de supporter la diversité et effectuer un vote doivent également être envisagées.

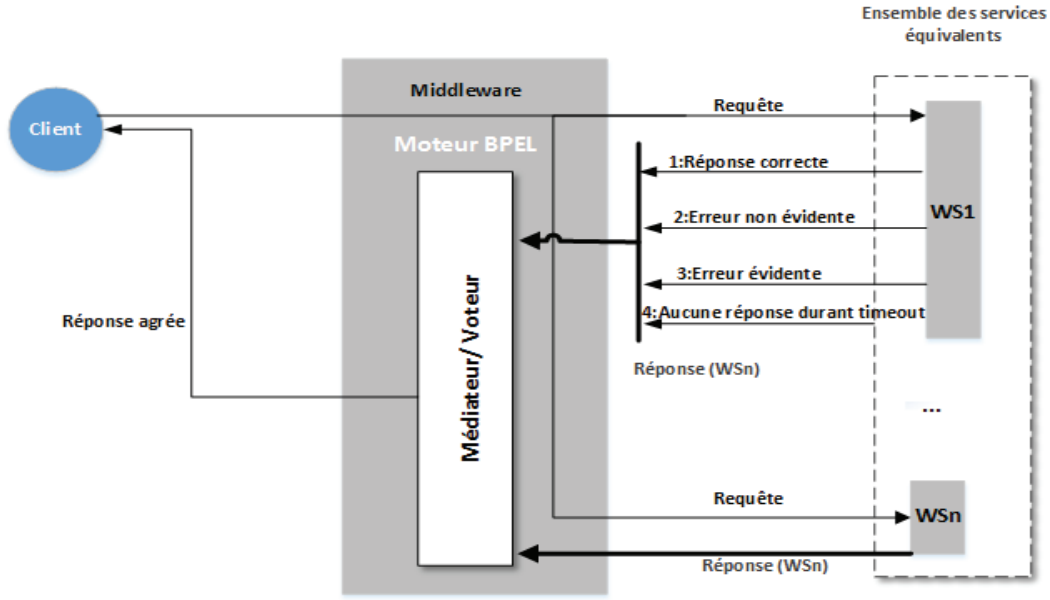


FIGURE 2.8 – **Architecture de composition verticale et horizontale**

2.3.2.4 WS-Mediator

Li and al. [CR08] proposent une architecture basée sur le concept de `WS-Mediator` pour supporter la résilience explicite et l'intégration dynamique des services Web. L'expression `résilience explicite` désigne l'utilisation explicite de l'information (métadonnées) sur les caractéristiques des composants d'un système pour guider la prise de décision au moment de la conception ou de l'exécution. Dans les SOA le terme adresse spécifiquement des questions de sûreté de fonctionnement pour garantir des applications hautement disponible [AoNuTCS07].

Le `WS-Mediator` consiste à un ensemble du sous-médiateur (`Sub-Mediator` en anglais) interconnectés entre eux formant une architecture de recouvrement (voir Fig. 2.9). Les `Sub-Mediators` sont globalement répartis sur l'Internet pour surveiller la fiabilité des services Web. Ils présentent les caractéristiques de sûreté de fonctionnement des services Web d'un point de vue client. Ils peuvent être implémentés de manière

identique ou différente. Le client invoque le `Sub-Mediator` comme le point d'entrée à l'architecture du `WS-Mediator`. Le `Sub-Mediator` intercepte l'interaction entre le client et le service composant et effectue le calcul de la résilience explicite et il applique une technique de tolérance aux fautes pour améliorer la fiabilité de la composition des services.

Les auteurs utilisent les techniques de diversité et de réplication des services Web. Le `WS-Mediator` est une solution déployée sur une infrastructure distribuée entre un ensemble de clients et de services Web. Le mécanisme de recouvrement se compose de plusieurs `sub-Mediators` fonctionnellement identiques et géographiquement répartis.

Dans le `WS-Mediator`, les `sub-Mediators` interceptent les appels des clients ou d'autres `sub-Mediators`. Chacun des `sub-Mediators` stocke les informations sur les services Web et les méta-données (l'adresse endpoint du service, les méthodes de liaison de message requis). Les méta-données représentent les caractéristiques de sûreté de fonctionnement des services Web, telles que le temps de réponse moyen, les principaux types de défaillances représentant leur comportement dans une base de données.

Les `sub-Mediators` surveillent les services Web à partir de différents localisations géographiques. Le `WS-Mediator` surveille les différents services Web avec leurs méta-données de fiabilité collectées et analysées par les différents `sub-Mediators`. Ces données contiennent le taux de disponibilité, le temps moyen de réponse, nombre du test, le nombre du test terminé avec succès. Elles sont utilisées pour sélectionner les services Web ; les services Web sont classés selon les valeurs de leurs méta-données.

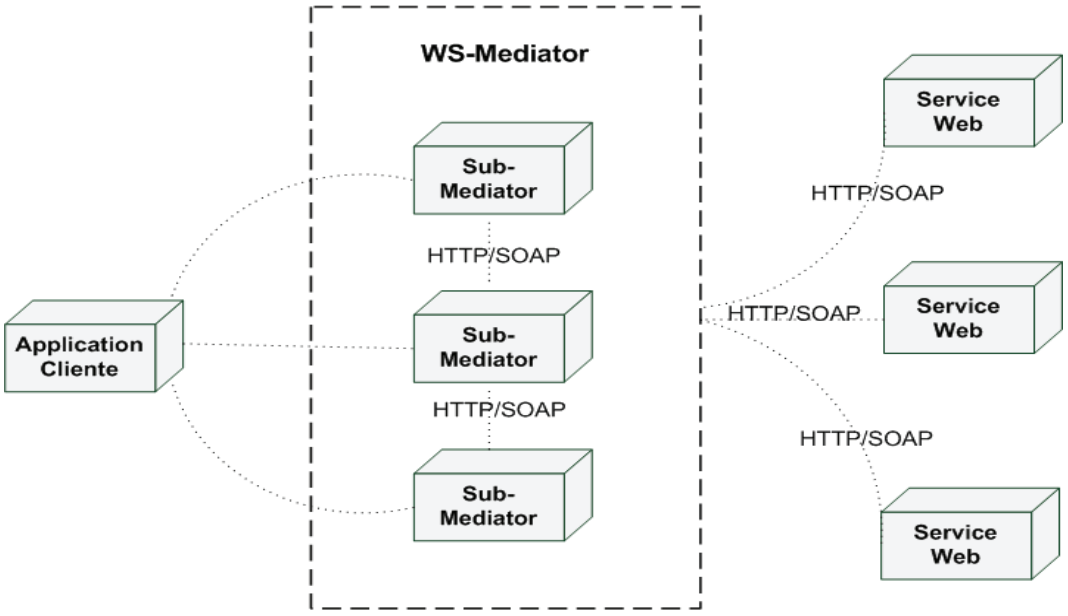


FIGURE 2.9 – **Architecture de Web service-Mediator**

2.3.3 Approches adaptatives de tolérance aux fautes

Dans la littérature, trois stratégies sont généralement proposées : i) dynamique, c.-à-d. sensible à la *QoS* ; ii) adaptative, c.-à-d. sensible au contexte ; et iii) combinaison de deux. D'après [DK10] la notion d'adaptabilité utilise le contexte pour opérer des changements structurelles et comportementales dans les applications tenant en compte différents environnements, différentes vues sur les données, ou encore diverses circonstances extérieures. Par la suite nous présentons plusieurs travaux de recherche se basant sur cette notion dans le domaine de la tolérance aux fautes.

2.3.3.1 Kalimucho

Louberry et al. [LRD11] s'appuient sur des applications à base de composants et de connecteurs (voie Fig 2.10). Chaque composant et le connecteur associé sont encapsulés dans un conteneur pour observer le contexte (e.g., QoS) et contrôler le cycle de vie de composant/connecteur (e.g., démarrage ou arrêt, migration). Les auteurs proposent une plate-forme distribuée, nommée Kalimucho. Cette plate-forme peut déclencher suite à des changements de contexte des modifications de structure et de déploiement de l'application via cinq services de base (ajout, suppression, connexion à un dispositif, déconnexion et migration). Kalimucho permet : (1) une capture du contexte (e.g., événements provenant de l'application) ; (2) une proposition de redéploiement (ou reconfiguration) fiable (i.e., avoir connaissance de tous les composants logiciels et des dispositifs disponibles et teste si le déploiement respecte les exigences QoS) ; (3) une gestion de l'hétérogénéité entre les dispositifs où seront déployés les composants de l'application. Kalimucho peut déployer ainsi une nouvelle configuration d'un service en évaluant un ensemble de configurations possibles respectant l'utilité et trouver un déploiement demandant le moindre changement dans le temps. Pour trouver un tel déploiement, il met en œuvre une heuristique pour le déploiement contextuel.

2.3.3.2 Modèle de réservation de services Web

Le travail présenté dans [SPJ11] met l'accent sur la conception des compositions de services Web robustes. Dans cette approche les auteurs proposent un mécanisme de réservation de services Web provenant de différents fournisseurs.

L'algorithme proposé permet aux consommateurs de services d'exécuter des processus d'affaires de manière fiable en présence de fortes contraintes de temps. Les services les plus critiques leur sont appliqués un mécanisme de redondance pour maximiser le profit obtenu lors de leur terminaison. Cet algorithme réserve les services pour une partie du processus d'affaire, afin de conserver une certaine souplesse en cas de défaillance (voir Fig 2.11).

Les auteurs utilisent différentes stratégies d'approvisionnement (parallèle, séquentielle, et hybride) pour réserver d'une manière automatique un nombre défini de ser-

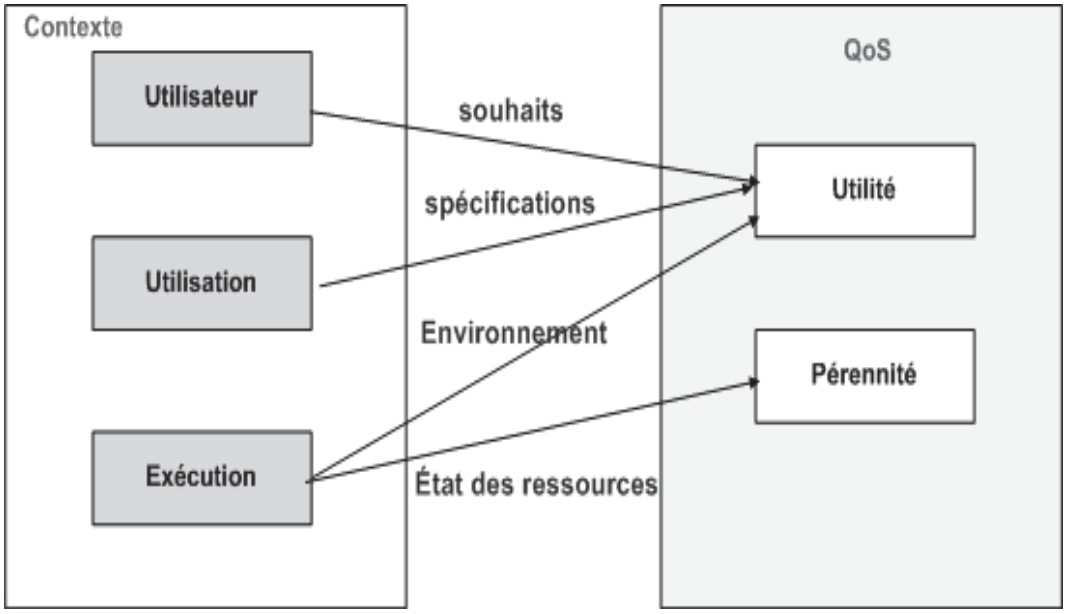


FIGURE 2.10 – **Interaction entre le contexte et la QoS**

vices Web pour chaque tâche. Ce nombre est d'abord défini par l'agent du système. Par contre, les services Web sont choisis de manière aléatoire à partir d'une liste de services capables d'accomplir une tâche donnée.

L'approvisionnement parallèle des services Web pour une tâche particulière permet au consommateur d'augmenter la probabilité globale du succès, ainsi que de diminuer la durée d'exécution de la tâche. Toutefois, l'introduction d'une telle redondance conduit aussi à un coût global plus élevé dans la mesure où tous les services Web doivent être payés.

Les auteurs prennent en considération plusieurs facteurs importants pour décider du nombre de services à réserver. Un grand degré de redondance est suggéré lorsque les services sont peu fiables. Ils équilibrent le coût de la redondance avec ses avantages. Quand les services sont chers, le degré de redondance sera ajusté en prenant moins de services. Cependant, quand les services ne sont pas chers, un plus grand nombre est proposé pour la stratégie parallèle. Les auteurs prennent aussi en compte l'importance de l'application pour décider combien dépenser pour les services et la redondance appropriée. Enfin, la structure de flux de travail est un autre facteur important. Par exemple, à la fin de la transaction les auteurs s'appuient sur des degrés plus élevés de redondance, afin de s'assurer que le grand investissement dans les services antérieurs ne soit pas perdu. Les auteurs introduisent aussi des modalités de pénalités pour les fournisseurs dans le cas où ils ne respectent pas leurs contrats de réservation.

Cette approche est importante dans le sens où elle adapte le degré de redondance en fonction de plusieurs facteurs clé. Cependant elle ne détermine pas quels sont les services à réserver, et sur quelle base ils sont choisis. Dans notre approche, une sélection des services Web les plus fiables est proposée, en fonction des exigences clients et de

l'historique d'exécution des services Web dans les différents groupes de diversité.

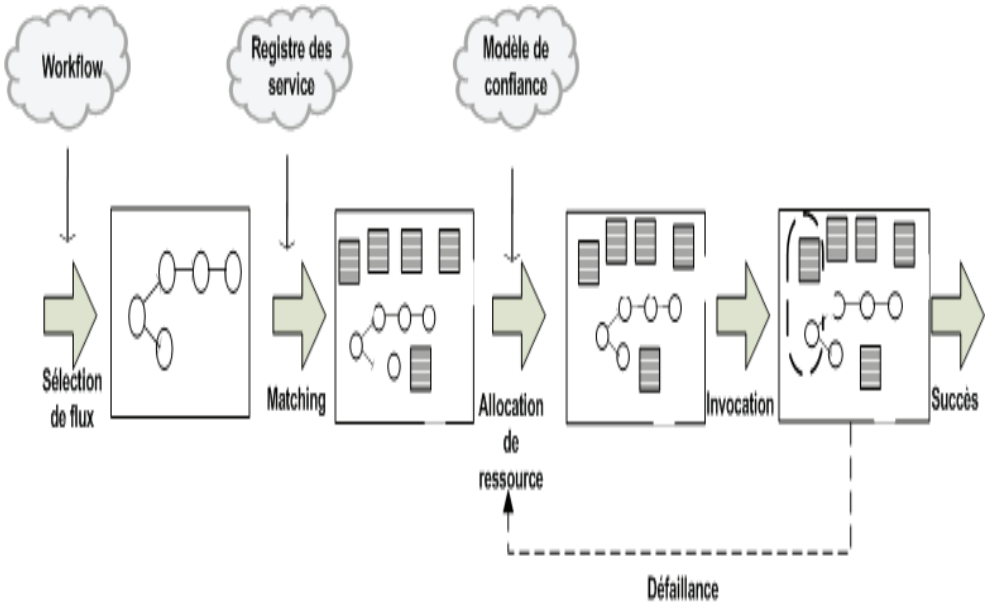


FIGURE 2.11 – **Cycle de vie d'un workflow**

2.3.3.3 Suitable Adaptation State (SAS)

Fabre and al. [FKP10] abordent la question cruciale d'adaptation des logiciels en ligne, consistant à déterminer si le système est dans un état adaptable ou pas. En effet, il s'agit d'établir les propriétés et les moyens de contrôler l'activité du système, pour atteindre un état dans lequel la modification de la configuration du logiciel peut être effectuée sans l'introduction d'un comportement incorrect (voir Fig 2.12). Pour cela, les auteurs définissent la notion de SAS (Suitable Adaptation State) représentant un état dans lequel la modification d'un composant peut être effectuée en toute sécurité.

Chaque composant individuel dans une configuration a son propre modèle réseau de Petri qui représente à la fois ses interactions avec d'autres composants et la causalité entre les tâches internes. L'approche proposée permet de décider quand le système est adaptable ou pas et d'orienter le système vers un état adaptable. Elle est appliquée à l'adaptation en ligne des mécanismes de tolérance aux fautes. En effet ces mécanismes sont considérés comme un ensemble de composants pouvant ainsi être adaptés en ligne selon certaines conditions d'utilisation particulières et d'évolution de la configuration du système.

L'adaptation de la tolérance aux fautes est souhaitable pour la maintenabilité du logiciel, mais aussi pour faire face aux changements dans le contexte opérationnel (e.g., environnement, hypothèses sur les fautes, manque de ressources, etc.). L'adaptation consiste à passer d'une stratégie à une autre en réponse à certains changements dans des conditions environnementales et des ressources.

Cette approche est basée sur l'étude de cas sur de mécanismes/middlewares de tolérance aux fautes où les conditions environnementales et de ressources permettent de décider si un switch vers une autre stratégie de tolérance aux fautes est nécessaire. Cependant, elle ne prend pas en considération le changement du contexte d'un point de vue de nombre de fautes observées. En réalité, une application tolérante aux fautes est définie en termes de nombre de répliques, variant selon le nombre de fautes et le type de fautes observées dans le système.

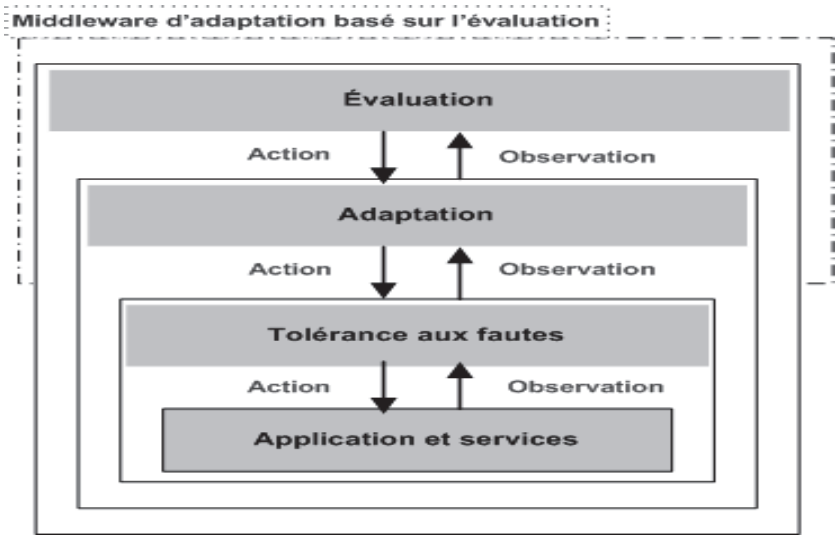


FIGURE 2.12 – **Overview de middelware d'adaptation**

2.4 Discussion

Parmi les nombreuses études visant à améliorer la sûreté de fonctionnement des services Web au niveau composant et composite, plusieurs approches se basent sur la réplication classique. Cependant, peu de travaux utilisent la diversité pour rendre les services Web composant et composite tolérants aux fautes. En règle générale, il est impossible de faire face à toute sorte de fautes dans une seule et unique solution. Par conséquent, les différentes approches sont basées sur des hypothèses de fautes à tolérer. La diversité est la meilleure solution pour éviter les fautes de mode commun pouvant persister dans des répliques identiques. Il existe deux façons d'appliquer la diversité des services, le protocole séquentiel tel qu'il est utilisé dans le *RecoveryBlocks* et le protocole parallèle tel qu'il est utilisé dans le $N - VersionProgramming$

Les stratégies de diversité ci-dessus ont été utilisées dans quelques solutions existantes. Contrairement aux mécanismes et protocoles de tolérance aux fautes basés sur la réplication qui sont bien formalisés et le contexte de leur application bien défini, ceux de la diversité restent un problème ouvert. De plus, les travaux existants manquent de

détails sur la sélection des protocoles adéquats pour tolérer un type particulier de fautes, ainsi que le nombre de services Web à diverses utilisés en tenant compte de la criticité de la fonctionnalité implémentée par les services.

En effet, il n'existe aucune garantie dans une composition de services Web que l'ensemble des services la composant soit fiable. Un degré de criticité doit être attribué pour mesurer la gravité de la défaillance sur la composition. Aussi, le choix du nombre des services Web à exécuter dépend de cette mesure. D'autre part, une mauvaise sélection des services Web conduit à de mauvais résultats. Une sélection des services appropriés améliore considérablement la sûreté de fonctionnement de l'ensemble de l'application. Par exemple, dans le protocole parallèle les services peuvent achever un consensus avec une valeur erronée si l'algorithme sélectionne les mauvais services similaires. Aussi, les travaux utilisant la diversité comme moyen pour tolérer les fautes, discutent rarement des stratégies de sélection.

Pour résumer, plusieurs problèmes subsistant encore dans la tolérance aux fautes dans les services composants et composites et non traités de manière satisfaisante :

1. La conception et la supervision du fonctionnement d'une application à base de services qui soit tolérante aux fautes et utilise la notion de diversité.
2. La détermination de nombre de répliques similaires pour chaque fonctionnalité, selon sa criticité.
3. La configuration de la diversité de services pour une meilleure sûreté de fonctionnement des applications à base de services. Plus précisément, comment identifier les meilleurs k services dans une diversité pour un protocole donné, comment les services interagissent entre eux (vote, ordonnancement).

2.5 Conclusion

La sûreté de fonctionnement des services Web est un domaine de recherche actif et important. L'architecture distribuée avec le couplage faible des services Web a apporté des avantages pour le développement d'applications distribuées plus flexibles et hétérogènes. Cependant, une telle architecture est par nature peu fiable. La recherche sur la sûreté de fonctionnement des applications à base de services Web doit faire face à deux types de défaillances à savoir les défaillances par crash et les défaillances Byzantines.

De nombreuses approches ont été développées pour assurer la fiabilité du service Web. Cependant, notre analyse montre que des limites de ces solutions empêchent leur efficacité. De nouvelles solutions sont nécessaires pour améliorer la fiabilité des applications de service Web du point de vue des clients afin de minimiser les problèmes causés par les défaillances des services. De nouvelles techniques sont également nécessaires pour améliorer l'efficacité de ces solutions en utilisant explicitement des stratégies de

diversité de services et en utilisant les services les plus fiables pour assurer des exécutions des services sûrs de fonctionnement.

Ayant motivé notre choix d'utilisation de la diversité comme une solution alternative à la réplication, les chapitres suivants présentent notre solution basée sur la diversité des services Web pour l'amélioration de leur fiabilité.

Chapitre 3

Services Web tolérants aux fautes : Une approche à base de diversité

Contents

3.1 Introduction

Le chapitre précédent discute les notions fondamentales des Services Web. Dans ce chapitre, nous proposons une approche pour rendre les services Web tolérants aux fautes. Cette approche exploite la diversité des services Web pouvant assurer une même fonctionnalité. Tous les composants logiciels y compris les Services Web sont sujets à des défaillances, celles-ci pouvant avoir des impacts importants sur tout le système (ou composition de services). Afin de garantir la continuité et le bon fonctionnement des applications à base de services en présence de défaillances, la réplication des services est souvent utilisée comme solution. Cependant la réplication présente deux principaux inconvénients. D'une part, les répliques étant identiques, elles sont alors automatiquement sujettes aux mêmes types de fautes de conception. Ces fautes se retrouvent dans toutes les répliques. D'autre part, la réplication est coûteuse en termes de consommation de ressources, elle implique un déploiement et une maintenance de toutes les répliques.

Pour remédier à ces inconvénients, nous proposons une approche de services web tolérants aux fautes basée sur la diversité de services. En effet, l'exploitation de services différents mais sémantiquement similaires remédie à la redondance de fautes de conception, et ne nécessite ni redéploiement ni maintenance des répliques.

Nous discutons dans ce chapitre la valeur ajoutée de la diversité pour assurer une exécution des services Web tolérants aux fautes. Les services Web qui offrent la même fonctionnalité sont regroupés au sein d'un même espace virtuel appelé groupe de diversité. Nous considérons ces services comme **sémantiquement équivalents** c'est-à-dire ils possèdent des descriptions *WSDL* différentes mais peuvent être considérés comme fournissant la même fonctionnalité. Ainsi, un groupe de diversité correspond à la réalisation d'une fonctionnalité.

En plus de la notion de groupe de diversité, des composants logiciels supplémentaires sont introduits dans ce chapitre à savoir :

- Un manager de services atomiques. Il permet de suivre l'état d'exécution d'un service pour signaler certaines éventuelles fautes.
- Un manager de groupes de diversité. Il permet de monitorer la réalisation d'un service abstrait via tout ou une partie des services concrets d'un groupe de diversité. Ce monitoring est mis en place via des protocoles spécifiques permettant la coordination des services d'un groupe pour remédier aux éventuelles fautes.
- Un manager de services composites. Il est utile dans le cas de compositions de services pour la gestion de plusieurs groupes de diversités.

Notre proposition se base sur des protocoles de tolérance aux fautes, spécifiques à la notion de diversité pour la gestion de groupes de diversité. Nous proposons deux types de protocoles à savoir un protocole pour des invocations séquentielles de services et un

protocole pour des invocations parallèles de services. Le premier protocole permet l'invocation d'un seul service à la fois, celui-ci est sélectionné sur un ou plusieurs critère(s) particulier(s). Si le service invoqué ne retourne aucun résultat (défaillance de type crash) ou retourne des résultats qui ne correspondent pas aux spécifications de l'application (défaillance de type Byzantine), un autre service est sélectionné dans le même groupe de diversité pour être invoqué. Ce processus continue jusqu'à l'obtention d'un résultat sans faute ou épuisement de services dans le groupe de diversité. Le deuxième type de protocole permet l'exécution concurrente de plusieurs services d'un même groupe. Ces derniers sont sélectionnés sur un ou plusieurs critère(s) particulier(s). Un seul résultat sera retourné au client ayant invoqué la fonctionnalité. Nous distinguons deux modes de sélection des résultats dans le protocole parallèle. Le premier mode est le parallèle sans vote et consiste à retourner le premier résultat qui correspond aux spécifications attendues. Le deuxième mode est le parallèle avec vote où le manager du groupe attend la réception d'une majorité de résultats pour ensuite effectuer un vote majoritaire (vote centralisé) pour sélectionner le bon résultat à retourner.

3.2 Définitions et Notations

L'approche que nous proposons est basée sur un certain nombre de définitions que nous décrivons ci-dessous :

Définition 3.1. ***Service concret :*** *Un service concret correspond à une fonctionnalité possédant une implémentation, et décrit dans un document WSDL. Il est accessible à travers une URI. Nous notons un service Web concret par sw.*

Exemple d'un service Web concret : Un exemple de service Web concret est le service amazon accessible à travers l'URI : $"http://aws.amazon.com/fr/"$ et possédant le document $WSDL :"http://webservices.amazon.com/AWSECommerceService/2005-03-23/US/AWSECommerceService.wsdl"$.

Définition 3.2. ***Service abstrait :*** *Un service abstrait correspond à une fonctionnalité mais sans aucune implémentation directe. Il n'a d'intérêt que s'il correspond à des services concrets. Nous notons dans la suite un service web abstrait par SWA.*

Exemple d'un service Web abstrait : Un exemple de service Web abstrait est le service de réservation de vol, celui-ci peut être implémenté par un service Web concret.

Définition 3.3. ***Groupe de diversité :*** *Un groupe de diversité correspond à un ensemble de services concrets qui implémente un même service abstrait. Ainsi, un groupe de diversité est défini par le couple* $< SWA_j, \{sw_1, sw_2, .., sw_n\} >$ *où* SWA_j *et* sw_i *(i=1,..,n) correspondent respectivement à un service abstrait et des services concrets. Nous notons un groupe de diversité par GD.*

Exemple d'un groupe de diversité : Un exemple de groupe de diversité est le couple <Réservation de vol, $\{SW - AirFrance, SW - Lufthansa, SW - BritishAirways\}$>. Il illustre le fait que la fonctionnalité abstraite de réservation de vol peut être concrétisée par un des trois services mentionnés.

La figure 3.1 schématise les trois notions de service concret, service abstrait et groupe de diversité.

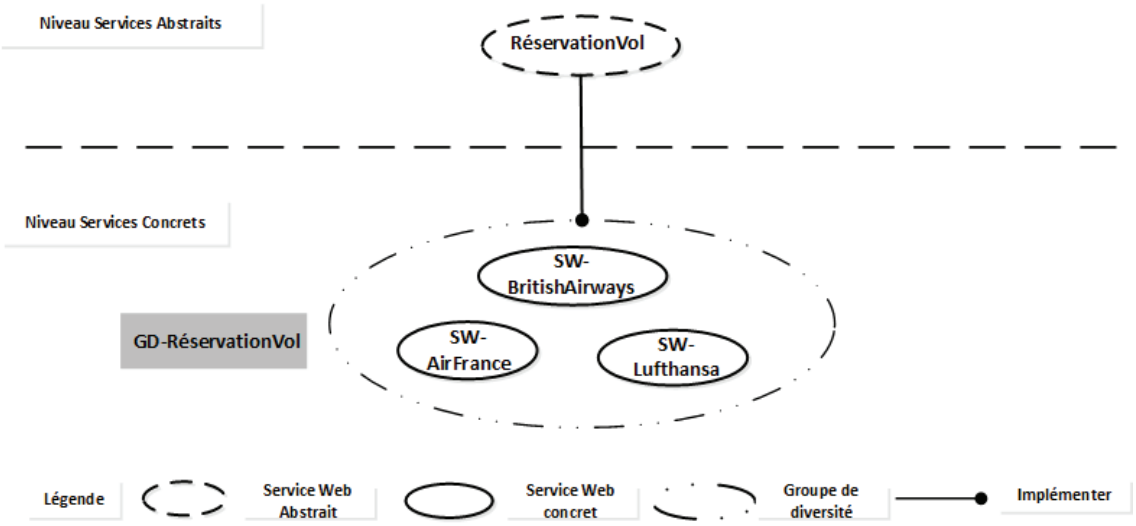


FIGURE 3.1 – **Exemple Service Web Abstrait implémenter par des services concrets**

Tous les membres d'un groupe ne participent pas toujours à la réalisation d'un service abstrait. La sélection des services concrets pour réaliser la fonctionnalité d'un groupe peut varier d'un contexte à un autre. Elle dépend de plusieurs paramètres dont notamment la criticité de la fonctionnalité. Plus la fonctionnalité est critique, plus le nombre de services concrets à impliquer est important.

3.3 Hypothèses

Un système ne défaille pas toujours de la même manière, ce qui conduit à la notion de mode de défaillance. Comme indiqué dans le chapitre précédent, quatre types de pannes (défaillances) ont été identifiés : les pannes franches (halte), les pannes par omission, les pannes temporelles et les pannes Byzantines. Dans notre approche nous considérons deux types de défaillances, les défaillances franches et les défaillances Byzantines.

3.3.1 Défaillance Byzantine

Une défaillance Byzantine se produit quand le résultat délivré par un service ne permet plus l'accomplissement de la fonction du système. Ceci se produit en général quand le résultat délivré ne respecte pas les spécifications attendues. Les défaillances Byzantines peuvent être malveillantes, c'est-à-dire produites de façon intentionnelle, ou non malveillantes quand elles sont produites sans aucun objectif de malice. Nous traitons dans notre approche les défaillances Byzantines non malveillantes. Nous considérons deux types de résultats :

- **Résultats incorrects** : Le service Web délivre des résultats qui ne correspondent pas à ceux attendus par l'utilisateur. Par exemple, un utilisateur peut s'attendre à recevoir un prix dans l'intervalle [50, 80], alors que le service Web lui renvoie un prix supérieur à 80. Cette situation correspond à une situation de résultat incorrect.
- **Résultats incomplets** : Le Service Web délivre des résultats partiels à cause des ressources limitées. Par exemple, un utilisateur peut s'attendre à recevoir la liste de tous les clients et le service lui en délivre qu'une partie. Un résultat incomplet peut se produire aussi lorsque seule une partie des outputs du service est retournée.

3.3.2 Spécification d'une composition de services Web

Nous considérons une composition de services Web comme une agrégation de services abstraits qui satisfait les demandes des clients. Les services abstraits correspondent aux fonctionnalités des groupes de diversités. Ils seront instanciés au moment de l'exécution de la composition et en tenant compte des aspects de tolérance aux fautes.

Exemple d'une composition des services Web :

Nous illustrons la composition de services Web par l'exemple classique d'un service de planification de voyages (Figure 3.2). Un client peut réserver un voyage de Paris à Londres, à condition de fournir la date de voyage et le numéro de la carte bancaire (paiement) à travers le site de l'agence de voyage.

L'agence de voyage est associée à quatre services Web abstraits où chaque *SWA* regroupe les services Web concrets *SW* qui assurent les mêmes fonctionnalités : **Réservation de vol**, **paiement**, **location de voitures** et **réservation d'hôtels**. Au moment de l'exécution, les fonctionnalités sont réalisées via des services Web concrets. La première fonctionnalité "Réservation de vol" peut être mise en œuvre par les services concrets suivants : $AirFranceWS$, $BritishAirwaysWS$, $LufthansaWS$. En parallèle avec cette fonctionnalité le client peut à travers la fonctionnalité réservation d'hôtel effectuer sa réservation, cette fonctionnalité est implémentée par les services concrets : $BestWesternWS$, $HolidayInnWS$.

Une fois le client a choisi son vol et son hôtel, la fonctionnalité de paiement en ligne permet au client d'effectuer le paiement à travers un des services concrets qui implé-

mentent cette fonctionnalité : *PayPalWS*, *GoogleCheckOutWS*, *OgoneWS*. Enfin, le client loue un moyen de transport entre l'aéroport et l'hôtel à travers la fonctionnalité location des voitures. Cette fonctionnalité est mise en œuvre à travers les services concrets : *CarRentalWS*, *EuropCarWS*

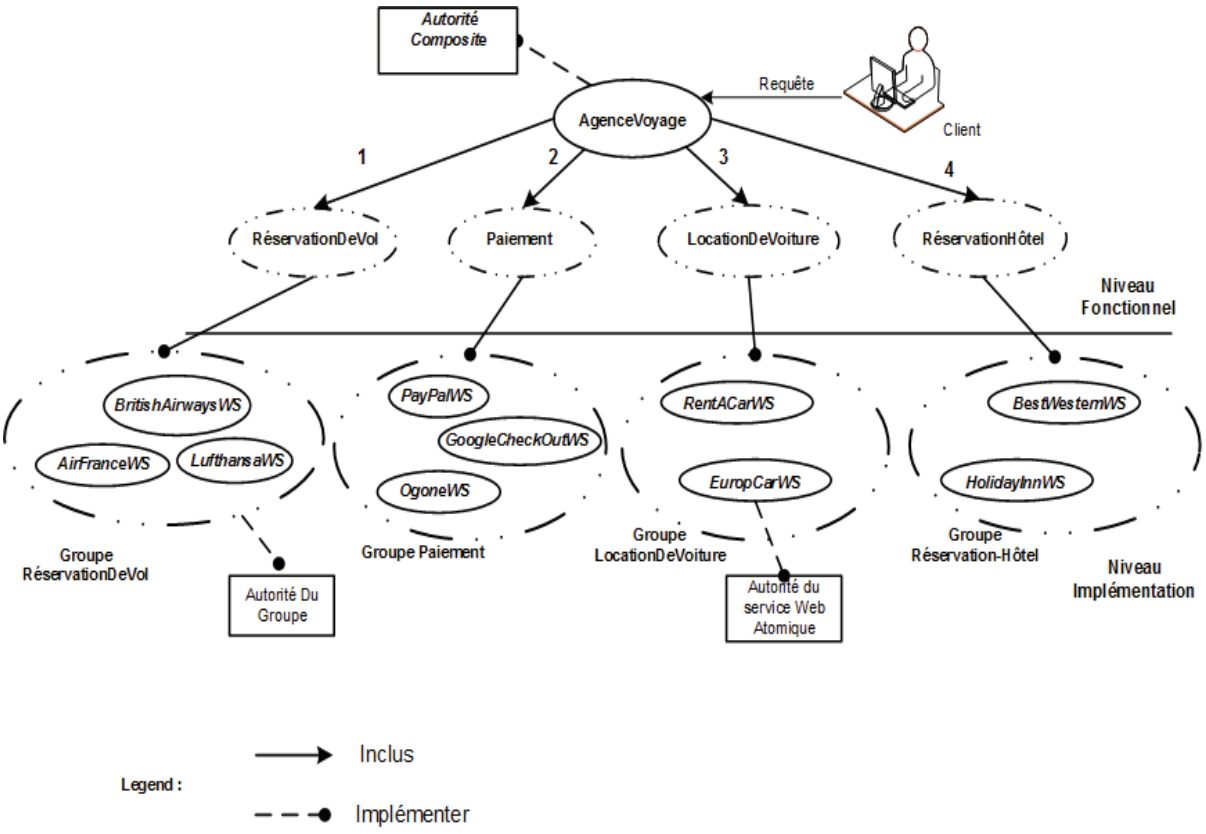


FIGURE 3.2 – **Exemple de service composite d'organisation de voyages en ligne**

3.3.3 Criticité des services abstraits

La conception d'applications à base de services qui soit tolérante aux fautes dépend du niveau de criticité de la fonctionnalité associée aux services abstraits. Le niveau de criticité mesure la gravité de la panne en cas de non réalisation de la fonctionnalité du service abstrait.

Les auteurs de [ESLH08] ont classé les applications en deux classes : applications sensibles et applications hautement disponibles. Dans le cas des applications sensibles telles que le contrôle du trafic aérien et la défense, les défaillances peuvent mettre en péril des vies humaines ou avoir un impact économique très élevé. Par conséquent, ces applications doivent fonctionner y compris en présence de fautes. Dans le cas des applications hautement disponibles comme dans les domaines de la banque et la télécommunication, les clients peuvent s'attendre à recevoir une réponse à leur demande, sans toutefois imposer des contraintes fortes comme sur le temps de réponse. Une ré-

ponse avec un certain délai de retard peut être tolérée.

Nous nous basons sur [ESLH08] pour identifier trois niveaux de criticité des services abstraits (donc des fonctionnalités) pour les besoins de la tolérance aux fautes :

- **Service abstrait critique** : Un service Web abstrait *SWA* est dit **critique** si l'exécution du groupe de diversité qui implémente la fonctionnalité de *SWA* **doit** se terminer dans un état de **Succès**.
- **Service abstrait semi-critique** : Un service Web abstrait *SWA* est dit **semi-critique** si l'exécution de groupe de diversité qui implémente la fonctionnalité de *SWA* **soit** se termine dans un état de **succès**, **soit** rend des **résultats incomplets**.
- **Service abstrait non-critique** : Un service Web abstrait est dit **non-critique** si l'exécution du groupe de diversité qui implémente la fonctionnalité de *SWA* peut se terminer dans n'importe quel état, et la défaillance du groupe **peut être ignorée** sans aucun impact sur l'application ou la composition de services.

Le choix de la criticité est déterminé selon les besoins des clients. Une fonctionnalité F_i peut être critique selon le point de vue d'un client C_1 et non critique selon le point de vue d'un autre client C_2.

3.4 Présentation de l'approche de tolérance aux fautes dans les services Web

3.4.1 Description générale de l'approche

La figure 3.3 illustre une vue globale de notre approche de tolérance aux fautes des applications à base de services. Elle inclut tous les concepts nécessaires pour garantir le bon fonctionnement des services Web.

Nous définissons trois niveaux d'abstractions : niveau composite, niveau groupe de diversité et niveau services concrets. Le niveau services concrets fait référence à tous les services pouvant être mis à disposition des utilisateurs. Cet ensemble de services peut évoluer dans le temps. Le niveau groupe de diversité est constitué d'un nombre quelconque de groupes, chacun défini par un service abstrait et un ensemble de services concrets similaires. Le nombre de services concrets d'un groupe peut évoluer dans le temps. Le niveau composite est dédié à la spécification d'une composition de services en termes de services abstraits.

La figure 3.3 montre également des entités additionnelles pour la gestion des trois niveaux introduits dans nos articles [FAMB11, AFMB12] : le manager du composite (*MCS*), le manager du groupe de diversité (*MGD*) et le manager du service Web concret

(*MSW*). Nous décrivons ci-dessous le rôle de ces différents managers :

1. **Manager du service Web concret (*MSW*)**. Il surveille l'exécution d'un service Web et signale au manager du groupe l'échec ou la réussite du service. L'échec d'exécution peut résulter de la détection d'un crash ou de la détection d'une faute byzantine en cas de non conformité du résultat par rapport aux spécifications attendues.

2. **Manager du groupe de diversité (*MGD*)**. Ce manager surveille la réalisation de la fonctionnalité du service Web abstrait à travers la mise en œuvre des protocoles d'exécution séquentiel et parallèle. Il est capable de détecter des fautes byzantines dans le cas d'absence de spécifications des résultats attendus. Ce manager transmet au manager de la composition l'état de l'exécution du service abstrait (Succès, Échec, etc.) et/ou le résultat de l'exécution du service.

3. **Manager du service composite (*MCS*)**. Il a pour rôle la gestion de la composition. Il est doté de fonctionnalités d'invocation des groupes de diversité et d'orchestration des services en tenant compte d'éventuelles fautes détectées par les managers de groupes et de la criticité des services abstraits.

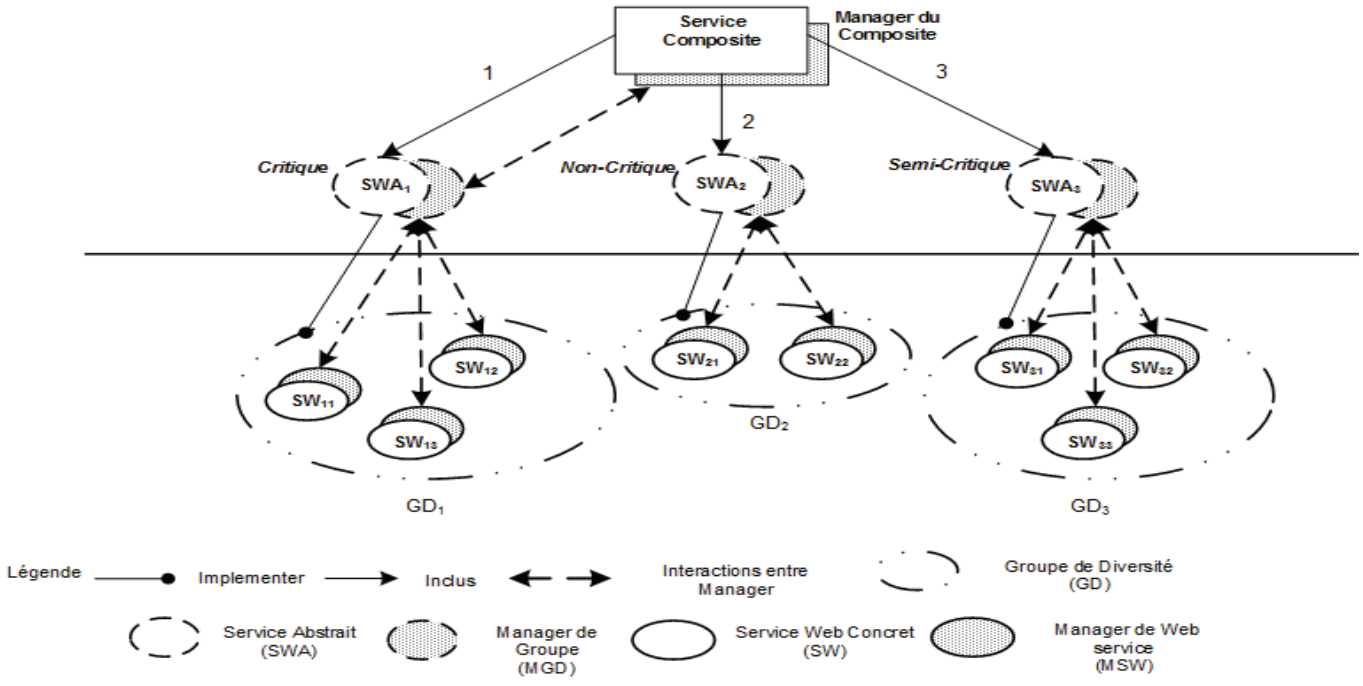


FIGURE 3.3 – **Vue Globale de l'approche de TF dans les services Web**

3.5 Modèle général de description des comportements et des intéractions des managers

3.5.1 Modèle général de description des comportements des managers

En vue de décrire le comportement des différents managers de l'approche proposée, nous présentons d'abord ci-dessous un modèle de description de comportements. Celui-ci est basé sur les automates à états finis. Notre choix est justifié par le fait que les modèles d'automates à états finis peuvent servir à déceler des comportements anormaux dans différents systèmes en décrivant la progression dans le temps.

Un automate à états finis est une machine abstraite et constitue un outil fondamental en mathématiques discrètes et en informatique. Il est représenté par un graphe orienté dont les nœuds représentent les états et les arcs, annotés par des éléments de l'alphabet, décrivent les transitions [WSWW06]. Il est utilisé dans la modélisation et contrôle des processus, et des protocoles de communication, ainsi que dans l'étude des langages formels.

Le modèle que nous décrivons se base sur les concepts états, évènements et actions qui peuvent survenir dans le comportement d'un service Web.

La description des comportements des managers suppose que :

1. Le modèle possède un état initial particulier.
2. Le comportement d'un manager doit avoir un ensemble fini d'entrées et/ou des événements qui peuvent déclencher des transitions entre les états.
3. Le comportement d'un manager à un instant donné t dépend de son état courant et des événements qui se produisent à cet instant.
4. Le comportement doit être défini pour chaque entrée ou évènement possible et ceci pour n'importe quel état.

Définition 3.4. ***Automate à états finis :*** *(finite state machine (FSM)). Un automate à états finis est un 5-uplet (Q, Σ, q_0, T, F) tel que :*

- *Q= {q_0, q_1, ..., q_n} : un ensemble fini d'états.*
- *Σ=(X_1, X_2, ..., X_m) : un alphabet (un ensemble de symboles).*
- *q_0 : un état initial.*
- *F={q_f, ..., q_s} : un ensemble d'états finaux.*
- *T= {t_0, t_1, ..., t_l} : un ensemble de transitions où chaque transition t_i est de la forme (q_a, X_k, q_b).*

Dans notre contexte de modélisation du comportement des managers, les automates à états finis capturent l'idée qu'à tout moment, le manager est dans un état particulier lorsqu'il répond à un évènement associé dans certains cas à des conditions. L'ensemble des états varient d'un manager à un autre. Les états communs à tous les comportements sont :

– État initial : {Initial}.
– États finaux : {Succès, Échec}.
– États intermédiaires : {Activé}.

Définition 3.5. ***Transition :*** *Une transaction T représente le passage d'un état à un autre. Elle est décrite par un triplet* $T = < q_i, \overset{alp}{\rightarrow}, q_j >$ *où :*
q_i *est l'ancien état,* q_j *est le nouvel état et* $\overset{alp}{\rightarrow} \in \Sigma$*, un alphabet permettant de représenter l'évènement déclencheur de la transition, les conditions du déclenchement et les actions pouvant être exécutées à l'issue de la transition.*

L'alphabet $\overset{alp}{\rightarrow}$ est de la forme : **Évènement [Condition] / Action**, où :
– **Évènements** déclenchent une activation possible de la transition.
– **Conditions** doivent être vérifiées avant de pouvoir activer réellement la transition.
– **Actions** représentent toutes les opérations pouvant être exécutées comme par exemple l'affectation de variables, les envois de signaux, la remise à zéro d'une horloge, etc.

Conditions : Les conditions peuvent être liées par des opérateurs de conjonction "(&&)" et de disjonction "(|)", la négation d'une condition est exprimée à l'aide de l'opérateur "(!)".

Exemple : Soit C_1 et C_2 deux conditions, la conjonction, disjonction et négation sont exprimées comme suit :
– C_1 && C_2
– $C_1 \mid C_2$
– ! C_1

Actions : Les actions alternatives sont représentées à l'aide de l'opérateur "(||)"

Exemple : $\text{Action}_1 \parallel \text{Action}_2$.

3.5.2 Communication entre les managers : Fonctions d'interactions

Les interactions entre les différents managers sont nécessaires pour la mise en œuvre de notre approche. En effet, lors de l'appel d'un service abstrait au niveau d'une application ou d'une composition, le manager de composition interagit avec le manager du groupe de diversité associé au service abstrait en question. Le manager du groupe de diversité interagit à son tour avec le ou les managers des services concrets pour une activation en séquentiel ou en parallèle des services. Les managers de services concrets remontent l'état de leurs services au manager du groupe de diversité. Ce dernier à son tour remonte au manager de la composition l'état du groupe de diversité. La figure (Figure 3.4) illustre les interactions descendantes du niveau service composite au niveau service concret et les interactions ascendantes du niveau service concret vers le niveau service composite. L'invocation d'un service abstrait provoque l'apparition de l'évènement **Appeler ()**. Celui-ci se produit entre le client et le manager du composite *MCS*.

Trois types d'interactions descendantes du composite vers le service Web concret sont définies :

- **Invoquer ()** : Cette fonction étiquette une interaction entre le manager du composite *MCS* et le manager d'un groupe de diversité *MGD*. Elle déclenche l'exécution du groupe.
- **ActiveP ()** : Cette fonction représente une action qui répond à l'évènement **Invoquer()**. Elle active en mode parallèle des services Web après le mappage des opérations et les paramètres des services Web.
- **ActiveS ()** : Cette fonction représente une action qui répond à l'évènement **Invoquer ()**. Elle active en mode séquentiel des services Web. Un seul service est activé à la fois et quand ce service Web tombe en panne, un autre service sera activé.

Ces fonctions seront détaillées dans la section 3.7.

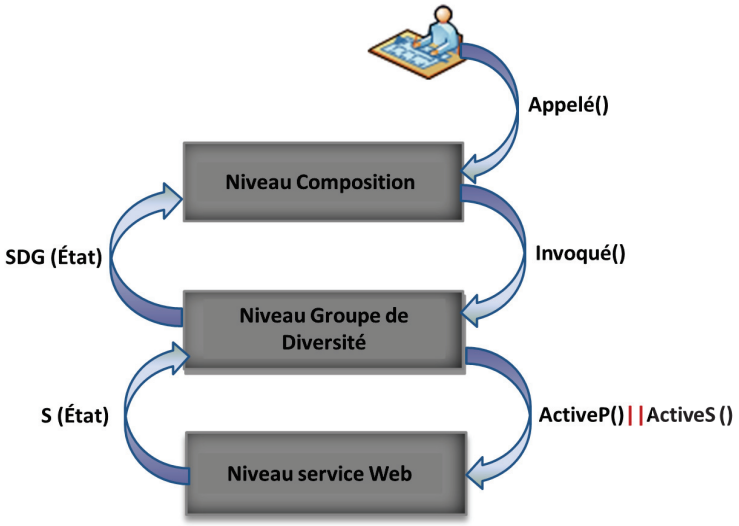


FIGURE 3.4 – **Interactions entre les 3 niveaux d'abstraction**

Deux types d'interactions ascendantes (bottom-up) entre les trois niveaux d'abstraction sont définis :

- **S(État)** : Cette fonction étiquette une interaction entre le manager du service Web *MSW* et le manager du groupe *MGD*. Elle transfère et signale l'état du service Web atomique au manager *MGD*. Cet état peut prendre cinq valeurs possibles : Succès, Échec, Ignoré, Succès-partiel et Annulé. Nous rappelons qu'au niveau du *MSW*, celui-ci ne peut détecter que les pannes de type crash s'il ne possède pas une spécification sur le résultat attendu. Par contre, si celui ci possède une spécification concernant le résultat attendu, il peut détecter les pannes Byzantines.
- **SGD (État)** : Cette fonction étiquette une interaction entre le manager *MGD* et le manager *MCS*, elle transfère et signale l'état du groupe de diversité au manager *MCS*. Cet état peut prendre trois valeurs possibles : Succès, Échec ou Ignoré.

Dans notre modèle, ces fonctions représentent soit des actions, soit des évènements constituant l'alphabet qui étiquète les transitions. Ces fonctions étiquètent généralement une transition vers un état final, sauf dans les cas où le *MGD* attend une réponse d'un autre service déjà invoqué.

3.6 Comportement des managers de services concrets et des services composites

3.6.1 Comportement du Manager du service composite (*MCS*)

Le Manager du Service Web Composite (*MCS*) est un composant logiciel, responsable de la gestion de l'application composite. Il agit comme un intermédiaire entre le client et l'application composite. Il joue le rôle de l'orchestrateur des différents Services Web abstraits implémentés par les groupes de diversité (*GD*). Le *MCS* se base sur la criticité de la fonctionnalité pour prendre une décision concernant le groupe de diversité défaillant. La figure 3.5 décrit le comportement du manager *MCS*

Un groupe de diversité est dit défaillant si l'exécution de tous les services Web concrets constituant ce groupe aboutit à un échec, ou les services Web au sein du groupe n'arrivent pas à approuver une valeur pour la réponse dans le cas de vote (protocole parallèle).

Lors d'une défaillance d'u groupe, le manager *MCS* peut prendre trois mesures possibles en fonction de la criticité de la fonctionnalité associée au groupe en question :

- Annuler (abort) : Le *MCS* annule l'exécution globale du service Web composite pour cause de défaillance d'un groupe critique.
- Ignorer (ignore) : Le *MCS* ignore le résultat fourni par le groupe de diversité défaillant. Cette mesure peut être prise quand le groupe de diversité défaillant est non critique.
- Compenser (compensate) : Le *MCS* défait l'exécution des services et reprend cette exécution à partir d'un point antérieur dans le flux de contrôle du service composite.

Etats d'un manager de services composites. Nous identifions quatre états possibles pour le comportement d'un manager de services composites *MCS* :

- État initial : {Initial}.
- États finaux : {Succès, Échec }.
- États intermédiaires : {Activé}.

Transitions : Les transitions apparaissent pour répondre à un évènement donné quand les conditions **Conditions** sont satisfaites. Le Tableau 3.1 représente l'alphabet étiquètant les transitions dans le *MCS*.

TABLE 3.1 – Description des transitions dans le manager de service Composite

Évènements	**Description**
Appeler()	Apparait quand le client envoie la requête au *MCS*
SDG (État)	Apparait quand le *MGD* signale l'état du groupe au manager de *MCS*
Conditions	**Description**
Suivant ()	Teste et vérifie s'il y' a encore des services Web abstraits à invoquer dans la composition
Actions	**Description**
Invoquer()	Déclenche l'exécution des groupes de diversité
Annuler()	Annule l'exécution du groupe de diversité
Sauter()	Ignore l'exécution d'un *GD* non critique
Valider()	Approuve le résultat retourné par *GD* en cours

Nous distinguons six transitions dans le modèle de *MCS*. Elles sont notées T_i = (État$_1$, Σ_i, État$_2$), tel que Σ_i est l'alphabet qui marque le passage de l'état **État**$_1$ à **État**$_2$ (i est le numéro de la transition).

Transition T_1 . Initialement, le manager *MCS* est dans l'état "initial". Dès qu'il reçoit une requête d'exécution d'un service abstrait, cette requête est tout de suite transférée au manager d'un groupe de diversité (MGD). Quand le *MCS* invoque le *GD* avec le protocole de *TF* approprié, il change son état et passe à l'état Activé. La transition déclenchée dans ce cas de figure est T_1 = (Initial, Σ_1, Activé). La première transition est T_1= (Initial, Σ_1, Activé) sachant que Σ_1= Appeler()/Invoquer().

Transitions T_2 à T_4 : Quand le manager *MCS* reçoit une réponse du manager d'un groupe de diversité, il peut alors réagir de trois manières différentes via les trois transitions suivantes :

T_i = (Activé, Σ_i, Activé), sachant que Σ_i peut prendre les valeurs suivantes :

- Σ_2 = SDG (État)[(État==Succès) | (État=Ignoré)&& Suivant()]/ Invoquer(). Cette transition est déclenchée si le *GD* termine son exécution avec **Succès** ou son **échec** peut être ignoré (fonctionnalité non critique). Le *MCS* peut demander l'exécution du prochain *GD* et reste à l'état **Activé** (**Transition T_2**).
- Σ_3 = SDG (État)[État=Agrée]/Sauté() | | Valider(). Cette transition est déclenchée quand le groupe de diversité *GD* approuve un résultat et envoie un évènement SDG(Agréé) au *MCS*. Le *MCS* peut soit approuver le résultat, soit ignorer le groupe tout en restant dans l'état **Activé** (**Transition T_3**).

– Σ_4 = SDG (État)[État=Insuccès]/Sauté() | | Annuler(). Cette transition est déclenchée quand un groupe de diversité *GD* rend un résultat partiel (**Transition T_4**).

Transition T_5 . T_5= (Activé, Σ_5 , Succès), sachant que Σ_5 = SDG(État)[(État = Succès) | (État=Ignoré | État= Insuccès)) && !Suivant()]. Cette transition est déclenchée quand le dernier *GD* termine son exécution avec succès ou son **Échec** peut être ignoré. Dans ce cas le *MCS* signale la réussite de la composition et par la suite il change son état à l'état Succès.

Transition T_6 . T_6= (Activé, Σ_6, Échec), avec Σ_6= SDG(État)[(État==Échec). Cette transition apparait si le *GD* ne retourne aucun résultat (crash) ou retourne des résultats incorrects (Byzantine). Le *MCS* peut en fonction de la criticité de la fonctionnalité :

– **Ignorer** la panne du groupe de diversité en question (fonctionnalité non critique) et **attendre** l'invocation du prochain groupe de diversité dans la composition. Dans ce cas le *MCS* persiste dans l'état **Activé**.
– **Annuler** l'exécution globale du service composite si le groupe défaillant est critique. Le *MCS* change alors son état à l'état **Échec**.

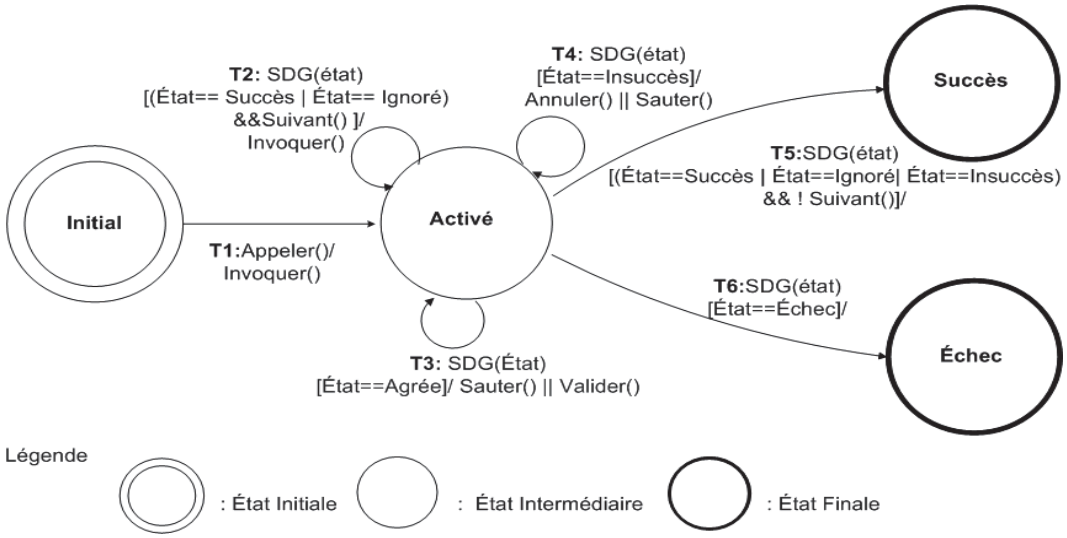


FIGURE 3.5 – **Modèle d'exécution de comportement du manager du composite** *MCS*

3.6.2 Manager du service Web concret (*MSW*)

Le manager du service Web concret *MSW* est en charge d'observer l'état du service Web et de rapporter l'état de succès ou d'échec au manager du groupe *MGD*. La figure 3.6 décrit le comportement du manager *MSW*. Nous identifions huit états possibles pour le *MSW* :

– État initial : {Initial}.

– États finaux : {Échec, Succès, Annulé, Ignoré, Succès-Partiel}.
– États intermédiaires : {Activé, Terminé}.

Les transitions : En plus des évènements définis dans les sections précédentes, le tableau 3.2 donne une description des différents évènements, conditions et actions à prendre en compte au niveau du manager du service concret *MWS*.

TABLE 3.2 – Description des transitions dans le manager de service Web

Évènements	**Description**
CrashServeur()	Apparait quand le serveur crash
Indisponible()	Apparait quand le service est indisponible ou il refuse d'exécuter la requête
Activé()	Apparait quand le *MGD* demande l'exécution de *SW* en question
Terminé()	Apparait quand le *SW* termine son exécution
Conditions	**Description**
Incomplet ()	Vérifie si le résultat de *SW* est incomplet
SélecteWS ()	Vérifie si le service a été sélectionné par le *MGD*
Actions	**Description**
S(État)	Signale l'état du *SW* au *MGD*

Le manager *MSW* peut transmettre un des cinq états au manager de groupes *MGD* :

À l'état Initial, lorsque le *MGD* demande l'exécution d'un service Web SW, deux événements peuvent survenir :

– **Transition T_1 .** T_1 = (Initial, Σ_1, Activé) avec Σ_1= Activer()/ : Si le service Web accepte la demande, le *MSW* change l'état du SW à l'état Activé.
– **Transition T_2 .** T_2 = (Initial, Σ_2, Échec) avec Σ_2= Indisponible()/S(état) : Si le service *SW* refuse de traiter la demande, le *MSW* change l'état du service Web à l'état échec et signale cet état au manager *MGD*.

Quand le service *SW* passe à l'état Activé, trois évènements peuvent survenir :

– **Transition T_3 .** T_3 = (Activé, Σ_3, Annulé) avec Σ_3= Agréer()/ : Quand le *MGD* atteint une valeur agréée, il annule l'exécution des services Web qui sont en cours.
– **Transition T_4 .** T_4 = (Activé, Σ_4, Échec) avec Σ_4= CrashServeur()/S(État) : Quand le serveur ne répond pas (crash), le manager *MSW* transmet l'état Échec au manager *MGD*.
– **Transition T_5 .** T_5 = (Activé, Σ_5, Terminé) avec Σ_5= Terminer/ : Quand le service Web termine son exécution, le manager *MSW* change l'état du service *SW* à Terminé.

Quand l'état d'un service est Terminé, le manager *MSW* peut réagir de 4 manières différentes :

– **Transition T_6 .** T_6 = (Terminé, Σ_6, Échec), avec Σ_6= SDG(État)[état=Insuccès]/S(État) : Cette transition apparait quand le manager *MSW* reçoit un évènement SDG(Insuccès)

à partir de *MGD*. Le type de faute dans ce cas est Byzantine car le service a retourné un résultat mais ce résultat est évalué comme étant incorrect par le *MGD* (algorithme de vote, test par rapport à la spécification...etc).

- **Transition T_7 .** T_7 = (Terminé, Σ_7, Succès-Partiel), avec Σ_7= SélecteSW [Incomplet && Sélectionner()]/S(État) : Cette transition apparait quand le résultat est évalué comme étant incomplet, mais le service est sélectionné par l'autorité du groupe de diversité pour rendre son résultat. Ce type de résultat est accepté quand le groupe auquel appartient le service en question est non critique ou est semi-critique.
- **Transition T_8 .** T_8 = (Terminé, Σ_8, Ignoré), avec Σ_8= SélecteSW(URI)[!Incomplet | !Sélectionner()]/S(État) : Cette transition apparait quand le résultat est évalué comme étant incomplet et le service n'a pas été sélectionné par l'autorité du groupe de diversité pour rendre son résultat. Dans ce cas, le résultat de service Web en question est ignoré et le *MSW* envoie son état au *MGD*.
- **Transition T_9 .** T_9 = (Terminé, Σ_9, Succès), avec Σ_9= SélecteSW(État)[correct && Sélectionner()]/S(État) : Cette transition apparait quand le résultat retourné est correct. Le manager *MSW* envoie dans ce cas au manager *MGD* l'état Succès du service S et reste quand á lui dans Succès.

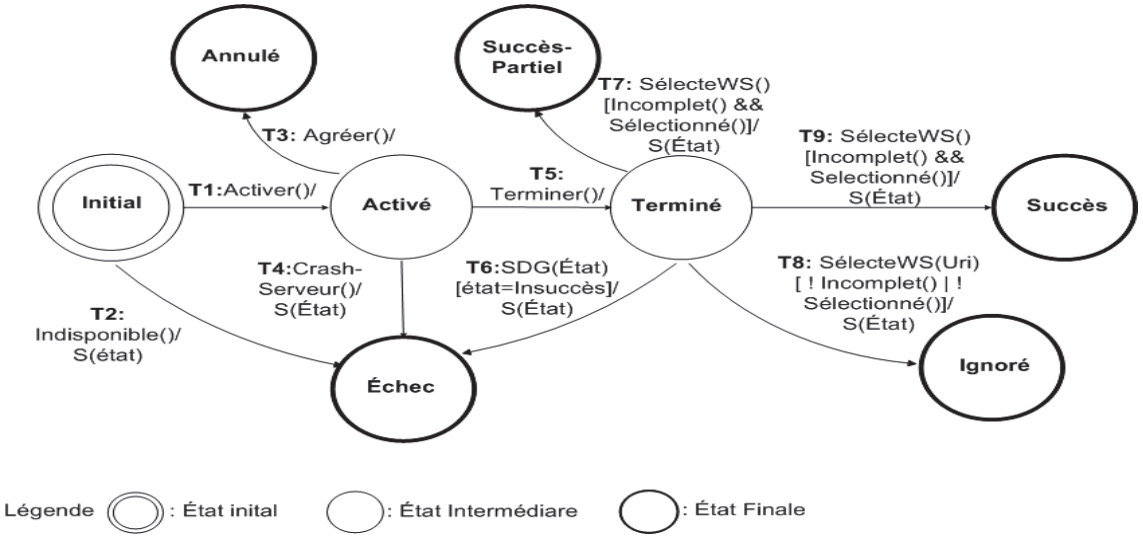


FIGURE 3.6 – **Modèle d'exécution du manager de service Web**

3.7 Manager du groupe de diversité *MGD*

Le manager de groupes de diversité *MGD* est en charge d'observer l'exécution des services Web et leurs interactions au sein d'un groupe et de rapporter l'état du groupe au manager de services composites. Le manager de groupes *MGD* peut transmettre au manager de services composites trois types d'états : **Succès**, **Échec** et **Ignoré** selon l'état

d'exécution du groupe et de la criticité de la fonctionnalité implémentée par ce groupe. Le type d'état varie selon le mode d'invocation et de communication des résultats. Le *MGD* est en mesure de :

- Invoquer les services Web sélectionnés selon la stratégie de recouvrement appropriée (Séquentiel, Parallèle).
- Signaler l'état du groupe au manager du composite.
- Communiquer le résultat au manager du service composite qui le transfère par la suite au client ayant invoqué l'application.

3.7.1 Stratégies de recouvrement du manager de groupes de diversité

Les stratégies de recouvrement sont les points prépondérants de notre solution. Une stratégie représente un mapping approprié entre l'état observé du service Web et l'action appropriée à entreprendre. La mise en place de stratégies de recouvrement est une tâche complexe qui nécessite de prendre en compte de nombreux problèmes. D'abord, un ensemble des services Web équivalents assurant la même fonctionnalité mais implémentés différemment doit être identifié et sélectionné. Ces services sont regroupés au sein du même espace virtuel appelé "**groupe de diversité**", mais seul un sous ensemble des services Web est sélectionné pour être exécuté. La sélection des membres du sous-groupe à invoquer sera discutée dans le chapitre suivant.
Dans ce qui suit nous discutons les problématiques qui concernent les stratégies de recouvrement elles-mêmes. Quelle est la stratégie à utiliser ? Quel est le nombre des services Web à sélectionner et invoquer dans le groupe ? Comment communiquer le résultat approprié (correct, cohérent et complet) au client ?

Les évènements peuvent être des fonctions d'interactions qui informent le manager *MGD* de l'état du service (c'est-à-dire un envoi d'état S(État) entre les managers *MSW* et le *MGD*). Cet état peut prendre les valeurs Succès, Échec ou d'autres types d'évènements qui seront détaillés par la suite. Les actions à entreprendre sont généralement une communication de l'état de groupe au *MCS* par la fonction (SGD(État)).

3.7.1.1 Protocole de Tolérance aux Fautes séquentiel

La stratégie d'activation séquentielle au sein d'un groupe de diversité implique d'envoyer la requête à un seul service Web qui la traite dans le groupe. Quand une erreur est détectée, un autre service dans le groupe sera activé pour exécuter la requête. Si le résultat du deuxième service ne satisfait pas la requête, un autre service sera sélectionné et exécuté. Le groupe est dit dans un état succès s'il arrive à un résultat correct, cohérent et complet selon le niveau de criticité exigé. Le groupe est dit défaillant si tous les services Web du groupe aboutissent à un échec (épuisement des services disponibles dans le groupe).
Le manager *MGD* est en charge de sélectionner un seul service Web et de l'invoquer. Ce service Web devient le leader du groupe. Quand ce leader est en panne, le manager du

groupe vérifie s'il existe un autre service dans le même groupe pour le sélectionner et le désigner comme un nouveau leader.

Les états du manager *MGD* **:** Différents états sont possibles pour le manager *MGD* :
- État initial : {Initial}.
- États finaux : {Succès, Échec, Ignoré}.
- États intermédiaires : {Activé, Insuccès}.

Les transitions : Les actions dans manager *MCS* au niveau de la composition de services deviennent des évènements dans le *MGD* (Invoquer(), Annuler(), Sauter()). Nous définissons dans le tableau suivant 3.3 les nouvelles conditions et actions utilisées pour représenter le comportement du manager *MGD* :

Table 3.3 – Description des transitions dans le manager du groupe pour le protocole Séquentiel

Conditions	**Description**
SW-Suivant ()	Vérifie s'il existe d'autres services Web à invoquer dans le *GD*
Actions	**Description**
ActiveS ()	Sélectionne un seul service Web à invoquer et si le service échoue, elle sélectionne un autre service dans le *GD*.
SDG(État)	Signale l'état du groupe au manager *MCS*

Le comportement du manager *MGD* quand il adopte le protocole séquentiel est décrit dans la figure 3.7. Nous distinguons les transitions suivantes :
Transition T_1. T_1 = (Initial, Σ_1, Activé) avec Σ_1= : Invoquer()/ActivateS() : Cette transition représente ce que le manager est censé faire lorsque la fonctionnalité qu'il représente est invoquée en mode séquentiel par le manager *MSC*. L'évènement Invoqué() déclenche l'exécution de groupe de diversité avec le mode d'activation séquentiel. L'action ActiveS () est déclenchée par le *MGD* afin de sélectionner un service Web dit leader (ou primaire) parmi les services disponibles dans le groupe. Le manager du groupe passe alors dans l'état Activé en attendant une réaction du manager des services web concrets.

À l'état **Activé** trois cas de figures sont possibles pour le manager *MGD* :

- **Transition T_2**. T_2 = (Activé, Σ_2, Activé) avec Σ_2 = S(État)[SW-Suivant()&& État == Échec]/ActivateS(). Cette transition fait rester le manager *MGD* dans un état Activé lorsque le service web leader échoue. En effet, dans ce cas, le manager *MGD* reste activé et vérifie s'il existe d'autres services Web à invoquer dans le groupe (fonction Suivant()). Si c'est le cas, il sélectionne un service et l'invoque.
- **Transition T_3**. T_3 = (Activé, Σ_3, Insuccès) avec Σ_3 = S(État)[!SW-Suivant()&& État == Échec]/ . Cette transition permet de passer le manager *MGD* dans l'état Insuccès lorsque tous les services web du groupe ont échoué.
- **Transition T_4**. T_4 = (Activé, Σ_4, Succès) avec Σ_4 = S(État)[État == Succès]/ SDG(État). Cette transition permet au manager *MGD* de passer dans l'état Succès lorsqu'un des services du groupe a réussi son exécution. Le manager *MGD* signale alors au manager de la composition le succès du groupe de diversité dans la l'exécution de la fonctionnalité.

Lorsque le manager *MGD* se retrouve dans l'état **Insuccès**, deux cas de figures sont possibles selon la criticité de la fonctionnalité :

- **Transition T_5**. T_5 = (Insuccès, Σ_5, Échec) avec Σ_5 = Annuler()/SDG(État). Si la fonctionnalité associée au groupe est **critique**, l'échec est inacceptable car son impact sur toute la composition est important. Dans une telle situation, le manager *MGD* signale l'état Échec au manager *MCS* par la fonction d'interaction SGD (État).
- **Transition T_6**. T_6 = (Insuccès, Σ_6, Succès) avec Σ_6 = S(État)[État == Succès-Partiel]/ SDG(État) : Cette transition est déclenchée si le groupe rend un résultat incomplet (un de ses membres rend un résultat incomplet). Le *MGD* signale alors un état "Succès" au manager *MCS*, ce dernier déclenche dans une transition vers **(1)** l'état "**Succès**" si le groupe en question est le dernier groupe dans la composition ou **(2)** reste dans l'état "**Activé**" s'il reste d'autres groupes à invoquer dans la composition.
- **Transition T_7**. T_7 = (Insuccès, Σ_6, Ignoré) avec Σ_7 = Sauter()/SDG(État). Cette transition est déclenchée si la fonctionnalité est **non critique**, l'échec est tolérable et la panne peut être ignorée. Dans ce cas le *MGD* rapporte l'état "ignoré" au *MCS*, ce dernier déclenche une transition vers l'état "**Succès**".

3.7.1.2 Protocole de Tolérance aux Fautes parallèle avec vote

Le rôle du manager *MGD* dans le cas d'un protocole parallèle est d'invoquer les services Web sélectionnés concurremment. Il collecte par la suite les différents résultats et en retourne un et un seul.

Concernant le protocole parallèle, seul un sous ensemble des services Web est sélectionné pour être invoqué concurremment. Le *MGD* envoie la requête aux membres du sous-groupe sélectionné pour exécuter la requête. Dans ce cas deux types d'approches

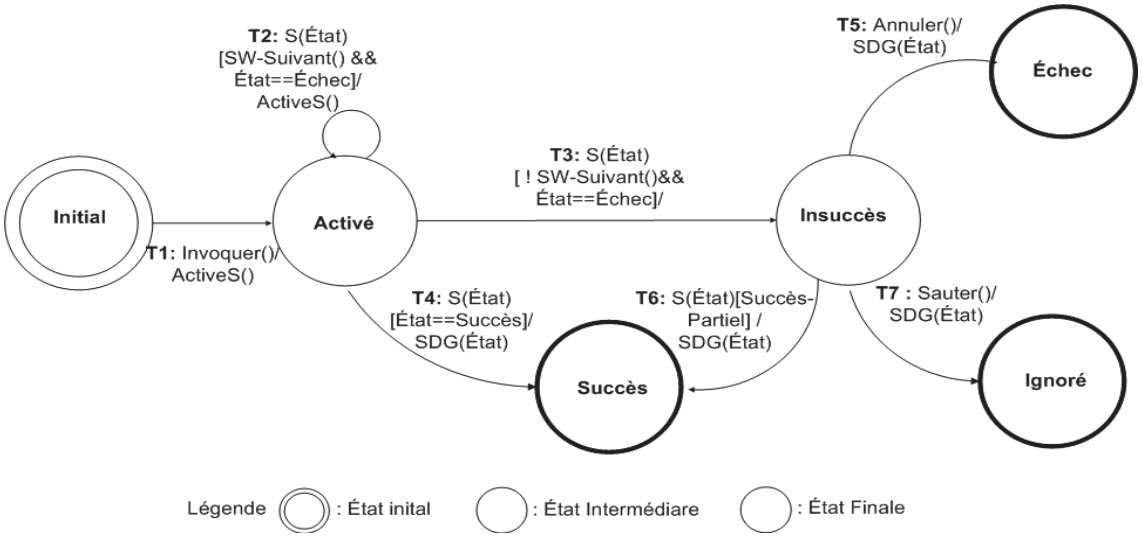


FIGURE 3.7 – **Modèle d'exécution du manager de groupe** *MGD* **dans le mode Séquentiel**

sont possibles pour communiquer le résultat :

- **avec vote** : le manager du groupe reste en attente jusqu'à ce qu'il reçoit les réponses fournies par n services sélectionnés dans le groupe. Lorsqu'il atteint une majorité des réponses, il effectue un vote majoritaire entre les réponses équivalentes. Ce mécanisme nécessite $2f + 1$ services pour tolérer f fautes de type Byzantine.
- **sans vote** : le *MGD* communique le résultat du premier service qui termine avec succès son exécution.

Le comportement du manager *MGD* dans le cas du protocole parallèle avec vote est décrit dans la figure 3.8. Nous identifions huit états dans la description du comportement du manager *MGD* dans l'activation parallèle avec vote :

- État initial : {Initial}.
- États finaux : {Succès, Échec, Ignoré}.
- États intermédiaires : {Activé, Insuccès, Terminé, Agrée}.

Les transitions : le tableau 3.4 suivant illustre les événements, conditions et actions additionnelles par rapport aux autres managers.

Transition T_1. T_1 = (Initial, Σ_1, Activé) avec Σ_1= : Invoquer()/ActivateP() : À l'état **Initial**, lorsque le *MCS* demande l'exécution d'un groupe *GD* (évènement Invoquer()) en mode parallèle avec vote, le *MGD* déclenche concurremment l'exécution des services Web par la fonction ActiveP (). Il change aussi son état à l'état Activé.

À l'état **Activé** le *MGD* peut réagir de deux façons différentes :

TABLE 3.4 – Description des transitions dans le manager du groupe pour le protocole Parallèle

Évènements	Description
SW-Terminé()	Apparait quand tous les services Web sélectionnés au sein du *GD* terminent leur exécution.
ÉchecSW()	Apparait quand tous les services Web dans le *GD* échouent.
Valider()	Apparait quand le *MCS* valide le résultat retourné par le *MGD*.
Conditions	**Description**
Agréer ()	Vérifie si le *MGD* arrive à un résultat agrée
Actions	**Description**
ActiveP()	Déclenche l'exécution des services Web sélectionnés concurremment.
Agréer()	Détermine la valeur approuvée entre les différentes valeurs des services Web.
SélecteSW()	Sélectionne un service Web parmi ceux qui ont retourné une valeur agréée pour continuer l'exécution.

- **Transition T_2**. T_2 = (Activé, Σ_2, Terminé) avec Σ_2= : SW-Terminé()/Agréer() : Quand une majorité des réponses des services activés est reçue (évènement Terminé()), le *MGD* déclenche le vote (la fonction Agréer ()) pour déterminer une valeur agréée. Il change ensuite son état à l'état Terminé.
- **Transition T_3**. T_3 = (Activé, Σ_3, Insuccès) avec Σ_3= ÉchecSW()/SDG(État) : Quand le service Web sélectionné ne répond pas (crash), le manager *MSW* signale au manager *MGD* la défaillance du service. Le manager change alors son état à l'état Insuccès.

À l'état **Terminé**, deux évènements peuvent survenir :

- **Transition T_4**. T_4 = (Terminé, Σ_4, Insuccès) avec Σ_4= [!Agréer()]/SDG(État) : Cette transition est déclenchée quand les services Web sélectionnés n'arrivent pas à approuver une valeur **!Agréé()**. Le manager *MGD* change son état à l'état Insuccès, et le rapporte au niveau du manager *MCS*.
- **Transition T_5**. T_5 = (Terminé, Σ_5, Agrée) avec Σ_5= [Agréer()]/SélecteSW()&& SDG(État) : Cette transition est déclenchée quand les services Web arrivent à déterminer une valeur approuvée en utilisant l'algorithme du vote. Le *MGD* change son état à l'état Agrée, et signale cet état à *MCS*.

Lorsque le manager *MGD* se trouve à l'état **Agréé**, deux cas de figure peuvent se présenter :

- **Transition T_6**. T_6 = (Agrée, Σ_6, Succès) avec Σ_6= Valider()/ SDG(État) : Cette transition est déclenchée quand le *MCS* valide le résultat du groupe (fonction Valider()). Le manager *MGD* change son état à l'état Succès et le communique au manager *MCS*.
- **Transition T_7**. T_7 = (Agrée, Σ_7, Ignoré) avec Σ_7= Sauter()/ SDG(État) : Cette transi-

tion est déclenchée quand le *MCS* ne valide pas le résultat approuvé par le groupe. Quand le groupe peut être ignoré, un évènement Ignorer() survient. *MGD* change son état à l'état Ignoré et le rapporte au *MCS*.

À l'état **Insuccès**, le *MGD* peut prendre deux mesures :

- **Transition T_8.** T_8 = (Insuccès, Σ_8, Échec) avec Σ_8= Annuler()/ SDG(État) : Cette transition est déclenchée en réponse à l'évènement annuler() déclenché par le *MCS*. Le *MGD* change son état à l'état Échec et le rapporte au *MCS* (fonctionnalité critique ou semi critique).
- **Transition T_9.** T_9 = (Insuccès, Σ_9, Ignoré) avec Σ_9= Sauter()/ SDG(État) : Cette transition est déclenchée quand la fonctionnalité est non critique. Le *MGD* change l'état de groupe à l'état Ignoré et rapporte cet état au *MCS*.

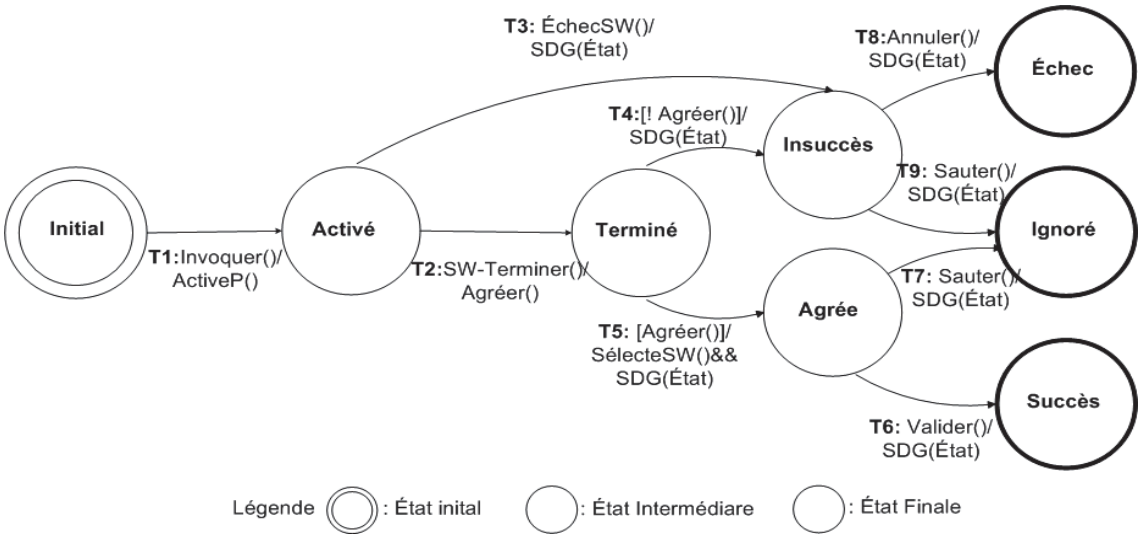


FIGURE 3.8 – **Modèle d'exécution du manager de groupe dans le mode Parallèle avec vote**

Algorithme de vote : Plusieurs approches ont été proposées dans la littérature [TCZH05, ZL10] pour mettre en œuvre un vote majoritaire entre un ensemble de services Web implémentant différemment la même fonctionnalité tout en possédant la même interface. Ils considèrent la valeur moyenne ou majoritaire des réponses comme correcte. Cependant, ils ne sont pas conçus pour traiter les services Web qui offrent la même fonctionnalité mais avec des interfaces différentes. Le manager du groupe *MGD* est doté de la fonction Agréé () qui implémente un algorithme capable de sélectionner la bonne réponse entre les différentes réponses qui proviennent des différents services Web équivalents.

Principe de l'algorithme *VDWS* **:** L'algorithme *VDWS* (Vote among Divers Web services) que nous proposons est décrit dans l'algorithme 1 et nécessite les entrées suivantes : un ensemble ordonné des résultats des services Web équivalents noté R et la déviation fonctionnelle acceptée entre les différentes réponses équivalentes notée δ. Cette

déviation est spécifiée dans l'expression des besoins par le client qui désire exécuter la fonctionnalité en question.

Nous décrivons les différentes étapes permettant de déterminer le plus grand ensemble $C_{largest}$ de services Web qui respecte cette déviation dans le sous-groupe de diversité. Cet algorithme partitionne le groupe sélectionné en grappes, en fonction du degré de matching D entre les différentes réponses données par l'algorithme de matching des différentes réponses. L'algorithme calcule ensuite le service Web racine (SW_{racine}) qui représente le centre de gravité de $C_{largest}$. Enfin, l'algorithme retourne la réponse produite par le service (SW_{racine}).

Algorithme 1: $VDWS$: Vote entre les résultats des Divers Services Web

Entrée : R = {Rep_{SW_1}, Rep_{SW_2}, ..., Rep_{SW_m}} and δ ;
Sortie : $Reponse - Agree$
1. Groupé Rep_{SW} in C={C_1, C_2, ..., C_p} sachant que :
$\forall$ Rep_{SW_i},Rep_{SW_j} $\in C_k$, $D(Rep_{SW_i},Rep_{SW_j}) \leq \delta$
2. Trouver $C_{largest} \in C$ sachant que :
$\forall\ i = 1, p \mid C_{largest} \mid \geq \mid C_i \mid$
3. Trouver $RepSW_{racine} \in C_{largest}$
4. $Valeur - Agree$ = $RepSW_{racine}$
5. Retourner $Valeur - Agree$.

Algorithme de matching : Selon [PKPS02] quatre niveaux de matching existent entre deux concepts *Concept*1 et *Concept*2 d'une même ontologie. Ces niveaux sont *Exact*, *Plugin*, *Subsume* et *Fail* :

- *Exact* : Si les deux concepts sont exactement les mêmes.
- *Plugin* : Si *Concept*1 représente un concept plus général que le *Concept*2.
- *Subsume* : Si *Concept*1 représente un concept plus spécifique que le *Concept*2.
- *Fail* : Si aucune des conditions ci-dessus n'est satisfaite.

Ces niveaux sont classés selon leur importance comme suit : Exact $\succ$ Plug in $\succ$ Subsume $\succ$ Fail, où $\succ$ est un opérateur qui mesure le niveau d'importance donné à un niveau de matching par rapport aux autres niveaux.

Le premier niveau *Exact* correspond au niveau le plus désiré par les demandeurs du service car les résultats retournés sont exactement les résultats attendus ou avec une petite déviation δ.

Le deuxième niveau *plugIn* représente un niveau désiré. Le résultat retourné dans ce niveau peut probablement être utilisé à la place de ce que le demandeur du service attend.

Le troisième niveau est le niveau *Subsume* puisque les exigences des demandeurs sont partiellement satisfaites. Ce niveau peut être acceptable dans le cas des fonctionnalités semi critiques.

Le dernier niveau est le niveau *Fail* où un résultat s'écarte des autres résultats. Ce niveau est tolérable dans le cas des fonctionnalités non critiques.

Algorithme 2: Algorithme de matching

Entrée : C_1, C_2, Nbr_{Exact},$Nbr_{Plug-in}$, Nbr_{Match}
Sortie : Nbr_{Exact},$Nbr_{Plug-in}$, Nbr_{Match}
Nbr_{Exact}=0 ; $Nbr_{Plug-in}$=0 ; Nbr_{Match}=0
1. Si *degreeMatching* (C_1, C_2)="Exact" **alors**
$Nbr_{Exact} \leftarrow Nbr_{Exact}$+1 ;
2. Si *degreeMatching* (C_1, C_2)="Plug-in" **alors**
$Nbr_{Plug-in} \leftarrow Nbr_{Plug-in}$+1 ;
3. $Nbr_{Match} \leftarrow$+1 ;
5. Retourner Nbr_{Exact},$Nbr_{Plug-in}$, Nbr_{Match} ;

3.8 Conclusion

Placés entre les clients et les fournisseurs, des composants logiciels, permettant d'assurer le bon fonctionnement des services fournis par ces prestataires, améliorent considérablement la confiance entre les deux acteurs de l'architecture basée sur des services Web.

Dans le cadre de notre approche, nous avons proposé trois composants logiciels. Ces composants sont dotés par des mécanismes de détection et de recouvrement d'erreur, qui assurent la continuité de l'exécution en présence de fautes.

Le premier composant *MCS* inséré entre le client et l'application joue le rôle d'intermédiaire entre les deux. Il est en mesure d'envoyer la requête de client, et d'invoquer le groupe de diversité. À la réception d'une réponse et en fonction de la criticité, le *MCS* peut soit valider ou annuler l'exécution globale en cas d'échec.

Le deuxième composant *MGD* monitore l'exécution des services au sein du groupe de diversité et les interactions entre eux. Il représente le cœur de notre système car il met en oeuvre des protocoles de tolérance aux fautes basés sur la diversité. Nous avons proposé deux types de protocoles : un protocole pour l'invocation séquentielle et un protocole pour l'invocation parallèle.

Le troisième composant est *MSW* surveille l'exécution d'un service Web et signale l'échec ou le succès d'un service Web atomique au *MGD*.

Nous avons défini aussi des fonctions d'interaction entre les différents niveaux de l'application composite. Ces fonctions gèrent le fonctionnement des différents services Web pour assurer la continuité de l'exéxution de l'application.

Chapitre 4

Sélection des services Web pour les besoins de tolérance aux fautes

Contents

4.1 Introduction

Le chapitre 3 a permis de mettre en évidence notre stratégie de tolérance aux fautes pour garantir une exécution fiable des services Web en utilisant la diversité comme solution alternative à la technique traditionnelle de la réplication. Nous avons défini des modèles d'exécutions des services Web en trois niveaux : composant, groupe de diversité et composite. Ces modèles sont supervisés par des managers contrôlant l'état d'exécution des services Web. Le manager du groupe est doté des protocoles de tolérance aux fautes pour coordonner les interactions des services Web au sein d'un groupe de diversité et garantir la fiabilité de ce dernier. Cependant, l'élaboration d'une stratégie de diversité requiert la définition de différents détails comme le nombre nécessaire de services Web divers, l'emplacement de ces services Web et les protocoles permettant de coordonner le fonctionnement du groupe de diversité. Ainsi, une mauvaise sélection des services Web peut conduire à un groupe de diversité avec une faible fiabilité ou à une défaillance du service fourni par ce groupe pouvant avoir un grand impact sur la composition.

Dans la plupart des approches existantes de diversité, les stratégies de diversité sont prédéfinies de manière statique au moment de la conception, ou encore utilisent tous les services Web disponibles dans le groupe de diversité. Cependant, ces approches ne sont pas toujours applicables dans des conditions dynamiques de l'environnement où l'exécution des services Web demande des ressources limitées et parfois coûteuses. Il est ainsi nécessaire de concevoir une approche de tolérance aux fautes adaptative pour répondre à différentes circonstances extérieures (i.e., changement de contexte). Le contexte peut, par exemple être défini par le nombre de fautes détectées, le type de ces fautes et le changement des exigences clients. Les approches adaptatives existantes sont intéressantes mais ne sont pas appropriées pour la diversité. En effet, les services Web peuvent provenir de différents fournisseurs conduisant ainsi à un problème de compatibilité entre les services Web et celui de confiance entre leurs fournisseurs.

De plus, comme les valeurs de Qualité de Service (QoS) (e.g., disponibilité et temps de réponse) requises pour le groupe de diversité peuvent varier d'une composition à une autre, seulement un sous-ensemble des services Web serait nécessaire. Ainsi, différentes configurations du groupe doivent être envisagées tenant en compte des facteurs tels que le degré de disponibilité et la déviation fonctionnelle acceptable.

Dans ce chapitre, nous soulevons les deux questions suivantes :

Q_1) : Quel est le degré de redondance nécessaire pour former et garantir le bon fonctionnement d'un groupe de diversité ?

Q_2) : Comment sélectionner les meilleurs services Web à impliquer dans le groupe de diversité selon le degré de redondance et la stratégie utilisée, et comment définir leurs ordonnancement dans une stratégie séquentielle ?

Ce chapitre est organisé comme suit. La première section présente les motivations pour la conception d'une nouvelle approche de sélection de services Web dans le groupe de diversité basée sur l'historique de l'exécution des services Web. La deuxième section

donne la vue globale de cette approche adaptative au changement de contexte en termes de nombre de fautes détectées et des services membres à impliquer dans le groupe de diversité. La troisième section discute du problème d'adapter la taille du groupe en fonction du nombre de fautes détectées dans les services et de sa solution. La quatrième section définie les données à collecter (échec, succès, exigences précédentes.. etc) pour construire l'historique d'exécution. Enfin la dernière section présente notre méthodologie pour identifier les services Web à impliquer dans le groupe de diversité en fonction de cet historique.

4.2 Motivations et Objectifs

Au moment d'invoquer des services Web dans le groupe de diversité, certains peuvent être indisponibles momentanément dû par exemple à des fautes de conception. D'autres peuvent quant à eux devenir indisponibles de manière permanente. Aussi, de nouveaux services Web peuvent rejoindre le groupe de diversité. De plus, le code et les ressources physiques d'un service Web peuvent être mis à jour sans aucune notification. La charge du trafic réseau et celle du serveur peuvent être aussi sujettes à des changements. Alors que le problème de non-disponibilité est adressé par la technique de diversité (de type N-Version Programming (NVP)), il reste néanmoins partiellement résolu car il dépend de la quantité de redondance utilisée et la disponibilité des services Web composants utilisés pour construire la composition de services [KL86].

La nature compositionnelle des services Web ainsi que la forte dynamique d'Internet rendent les stratégies de tolérance aux fautes classiques non pratiques dans un environnement du monde réel. En effet, ces dernières ont été traditionnellement appliquées à un ensemble immuable de services Web sémantiquement équivalents et sont agnostiques au contexte [LRD11]. Il devient alors évident que l'efficacité d'un groupe de diversité dépend de la disponibilité des services Web le composant. En effet, un vote majoritaire parmi ces services Web ne pourra pas garantir la disponibilité du service attendu de l'application si la majorité devient simultanément indisponible. Pour les applications temps-réel, tout délai additionnel dans le temps de réponse d'un service Web peut impacter l'efficacité du groupe de diversité à délivrer les résultats en temps voulu.

Toutes ces caractéristiques non prévisibles des services Web sont un véritable challenge lors de la détermination de la configuration du groupe appropriée pour assurer son bon fonctionnement. Pour cela, il y a un besoin urgent pour trouver des solutions dynamiques et adaptatives de tolérance aux fautes. Les rares travaux (e.g., [Bes10, FKP10]) traitant ce problème dans le domaine SOA proposent une gestion sophistiquée de la diversité en prenant en compte le contexte et la QoS. Des exemples d'informations contextuelles sont : le nombre actuel de services Web dans le groupe de diversité, l'évolution des résultats de vote et l'état opérationnel de chacun de ces services Web (e.g., la charge,

le temps d'exécution, et la disponibilité). Face à des changements de contexte, une stratégie dynamique et adaptative de diversité devrait être capable d'accorder le nombre de services Web nécessaires et de changer dynamiquement la sélection de ces services Web dans le groupe de telle manière à maintenir l'efficacité de la stratégie de diversité, en minimisant par exemple les effets négatifs de l'appel à ces services Web devenus inappropriés.

Ayant motivé les insuffisances des approches existantes, notre travail vise à définir une approche adaptative au changement du contexte, en termes du nombre des fautes détectées afin de sélectionner les services Web les plus fiables. La valeur ajoutée de notre approche par rapport aux approches existantes est de déterminer les services Web membres d'un groupe de diversité pour chaque stratégie. La sélection des services est basée sur les exigences des clients ainsi que sur l'historique d'exécution des services Web, ce qui permet d'avoir une idée sur leurs comportements dans les précédentes exécutions. Une méthode d'ordonnancement des services Web basée sur les items fréquents permet d'améliorer considérablement la configuration de notre groupe de diversité dans le cas d'une stratégie séquentielle est présentée.

4.3 Overview de l'approche adaptative au changement du contexte

Dans cette section, nous présentons les fondements sur lesquels se base notre nouvelle approche de diversité adaptative ainsi qu'une vue globale dans laquelle la configuration du groupe est construite dynamiquement en fonction de l'historique des interactions au sein du groupe et du contexte dans lequel le groupe est exploité. Cette approche permettra de satisfaire les exigences de tolérance aux fautes initialement définis par un ou plusieurs utilisateurs finaux.

4.3.1 Fondements de base

Pour répondre aux questions relatives au degré de redondance et à la sélection de services Web, notre approche se base sur les propriétés suivantes :

- **Historique d'exécution des services Web** (*Log*) : Le fichier *Log* contient toutes les données nécessaires pour comprendre le comportement d'un service Web dans les exécutions précédentes comme des données liées à la fiabilité telles que le succès et l'échec, des données liées aux comportements d'un service au sein du groupe de diversité ou encore le degré de confiance des services Web membres à un service dans le même groupe de diversité.
- **Changement du contexte** : La notion de contexte représente l'ensemble des circonstances dans lesquelles se produit un évènement. Une sensibilité au contexte permet de personnaliser et d'adapter les applications à différents environnements, à différentes vues sur les données, à diverses circonstances extérieures représen-

tant les aspects auxquels un service doit s'adapter. Dans notre travail, les aspects sont la variation du nombre de fautes détectées (i.e., augmentation ou diminution), le changement de type de fautes détectées.

– **Exigences clients** : Elles représentent les besoins fonctionnels en tolérance aux fautes des clients actuels, comme le taux de défaillance maximum toléré mais aussi des besoins non fonctionnels en termes de temps de réponse maximum toléré, coût des ressources accepté, etc.

4.3.2 Vue globale

Dans notre approche, le principe d'une stratégie adaptative sensible au contexte consiste en trois étapes :

– **Choix du protocole**. Cette étape permet de déterminer le protocole ou mode d'interaction (parallèle ou séquentiel) pour une re-configuration au moment de l'exécution. Le choix du protocole repose sur le contexte en termes de types de fautes observées, d'exigences de l'application vis à vis du temps de réponse et les ressources disponibles. Le protocole parallèle fournit un temps de recouvrement rapide plus spécifiquement dans le cas sans vote ; il est souvent dédié aux applications avec des contraintes temps réel. Le protocole parallèle avec vote est recommandé lorsque le taux d'échec est trop élevé ou encore lors de présence des fautes Byzantines ne pouvant être détectées sans le vote. Par contre, l'utilisation de ce protocole a un coût élevé en termes de consommation de ressources. Quant au protocole séquentiel, il consomme moins de ressources mais ne fournit pas le résultat aussi rapidement que le protocole parallèle sans vote.
  Le tableau suivant 4.1 récapitule les différents cas où l'utilisation d'un protocole est recommandée. Le symbole $\times\times$ signifie que l'utilisation d'un tel protocole en présence de certaines conditions est très recommandée. Par exemple si une application a une forte contrainte du coût, l'utilisation du protocole séquentiel sera le bon choix, comme il ne consomme pas beaucoup de ressources.

Table 4.1 – Choix de la stratégie de *TF* selon les différents critères exigés

Critères exigés / Protocoles	Byzantines sans spécification	Byzantines avec spécification	Crash	Temps	Coût
Séquentiel		×	×		× ×
Parallèle avec vote	× ×		×	×	
Parallèle sans vote	×	×	×	× ×	

– **Calcul du degré de redondance.** Cette phase permet de déterminer le degré approprié de redondance à utiliser pour le prochain tour de vote dans le protocole parallèle en tenant en compte les fautes observées dans les précédents tours et

de déterminer le degré de réservation du service dans un protocole séquentiel. Il s'agit donc d'évaluer le degré de diversité dépendant de l'état courant du contexte. D'une part, dans le cas d'une absence des perturbations exceptionnelles (i.e., nouvelles fautes détectées), la taille du groupe doit être réduite de manière à éviter l'utilisation inutile de services Web. D'autre part, lorsque le nombre de services Web initialement prévu devient insuffisant pour compenser les perturbations courantes, il serait avantageux de réviser de manière dynamique ce nombre incluant l'ajout de services Web si nécessaire.

– **Sélection des services Web divers**. Le processus de sélection de services Web doit assurer un compromis optimal entre les attributs de tolérance aux fautes (e.g., disponibilité et fiabilité) et les facteurs de performance (e.g., temps de réponse).

Ainsi, notre idée est d'exploiter l'historique des interactions des membres au sein d'un groupe pour déterminer quels sont les sous-groupes qui fonctionnent le mieux ensemble. Cela consiste à observer de manière continue tout changement dans l'état opérationnel des services Web disponibles dans le groupe de diversité et de collecter les informations contextuelles. On pourra ainsi identifier les services Web constituant une potentielle menace contre l'efficacité du groupe en entier pour les exclure.

4.3.3 Architecture

Notre approche de sélection des services Web fiables choisit à partir d'un ensemble des services Web SW={sw_1, sw_2,..., sw_m} équivalents disponibles (m est le nombre du services Web disponibles), un sous-ensemble des services formant le groupe de diversité GD_i={sw_{i1}, sw_{i2},..., sw_{in}} pour assurer le niveau de fiabilité et disponibilité désiré par le client (exprimé en termes d'exigences). Afin de constituer le groupe de diversité, nous devons déterminer le degré de redondance noté n des services Web à utiliser, dépendant du mode d'interaction.

La figure 4.1 montre les différentes étapes et les composants nécessaires à la mise en œuvre de notre approche adaptative de diversité basée sur l'historique des exécutions. Il s'agit dans un premier temps d'extraire les données nécessaires pour la sélection des services Web. Cette étape a pour objectif de collecter les différentes informations pertinentes à partir d'un fichier Log contenant les évènements observés au moment de l'exécution. Les informations collectées au cours des exécutions concernent aussi bien les services Web (e.g., leur fiabilité) que leurs interactions au sein du groupe (Section 4.5). L'objectif est de comprendre le comportement des services Web dans des exécutions précédentes en termes du nombre de fautes observées, du temps d'exécution des services Web, etc. La figure 4.1 montre aussi un module de génération pour extraire les données à partir du Log et enregistre les données pertinentes dans une table de monitoring. Cette table structure les données pour faciliter leur utilisation par la procédure de calcul des services Web à impliquer dans le groupe (Voir Section 4.5.4).

Dans un deuxième temps, il s'agit de déterminer le degré de redondance nécessaire

étant donné le protocole d'exécution (Parallèle ou Séquentiel), le type et le nombre de fautes détectées dans le groupe de diversité. La dernière étape consiste à utiliser les données collectées (e.g., succès ou échec, valeur ou crash, temps d'exécution, coût, exigences dans le passé, etc.), ainsi que les exigences actuelles d'un client, pour calculer à quel point le service Web a réussi les exigences. Ces exigences sont définies en termes de taux de défaillances maximum toléré, temps de réponse maximum toléré et le coût maximum toléré. Elles sont incluses dans la requête et capturées par le Groupe Manager afin de définir les membres du nouveau groupe.

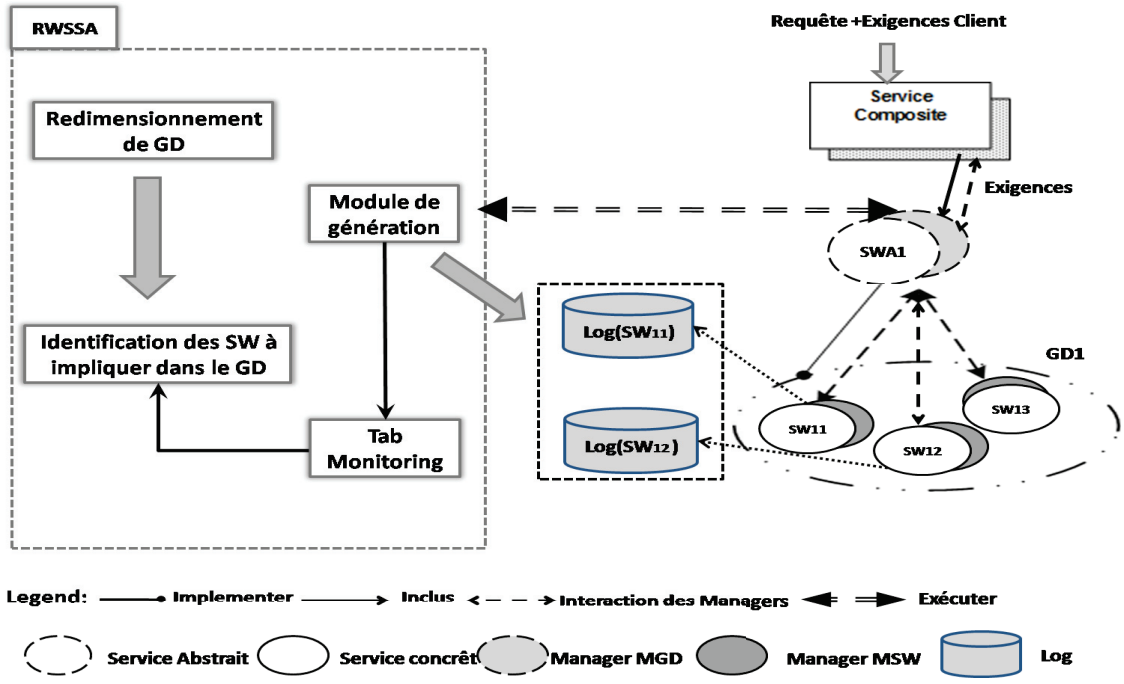


FIGURE 4.1 – **Enchaînement des étapes d'adaptation**

4.3.4 Spécification du problème

Dans cette section, nous allons d'abord définir les paramètres du problème de sélection tels que la pertinence d'un service Web, la qualité d'un groupe, et les exigences clients. Ensuite, nous montrons comment ces paramètres interviennent dans la formalisation de ce problème.

4.3.4.1 Pertinence d'un service Web

La pertinence d'un service Web au sein d'un groupe de diversité représente une mesure afin d'estimer l'impact de la défaillance de ce service sur le groupe de diversité. Par exemple, un service Web de paiement par carte bancaire est plus pertinent qu'un service Web de paiement par chèque dans un contexte donné. Pour déterminer la pertinence d'un service Web, nous identifions les principaux facteurs pouvant influencer

cette propriété comme suit :

- **Informations provenant du client**. $\mathrm{Rp}_c(\mathrm{sw}_{ij})$ représente la valeur de réputation donnée par un client c à un service Web sw_{ij} en termes de préférences pour la réalisation de la fonctionnalité demandée par rapport à un autre service. Par exemple, pour la fonctionnalité paiement un client préfère un paiement par carte bancaire à un paiement par chèque.
- **Informations liés au groupe de diversité**. Elles représentent :

  1. la charge d'un service (Q) en termes du nombre de requêtes en attente.

  2. le degré de confiance ($\mathrm{Cf}_{sw_{ij}}(sw)$) (j=1,...,n) : représente la valeur de confiance placé par les services Web sw_{ij} (j=1,...,n) membres du groupe de diversité GD_i à un service Web sw dans le même groupe GD_i. Lorsqu'un service Web membre d'un groupe de diversité GD_i juge un autre service dans le même groupe, il va d'abord utiliser sa propre expérience avec ce service c.-à-d. : - Est ce que les deux services ont participé à la même procédure de vote, consensus ? - Est ce que les deux services proviennent d'un même fournisseur, ou de deux fournisseurs qui ont un contrat de confiance entre eux ?

  3. le poids associé au degré de confiance (w) donné par un service Web.

Nous calculons le degré de confiance comme une agrégation de confiance des services Web dans le service sw, avec un poids plus important associé au degré de confiance d'un service Web pertinent par rapport au poids associé à un service Web moins pertinent dans le groupe.

Soit $\mathbb{SWP}$ l'ensemble des services Web pertinents dans le groupe de diversité (swp $\in \mathbb{SWP}$) et $\mathbb{SWR}$ l'ensemble des services Web non pertinents dans le GD_i. Le degré de confiance est donné par l'équation suivante :

$$Cf_{sw_{ij}}(sw) = \frac{1}{nb_{swp}} \times \sum_{p=1}^{nb_{swp}} (Cf_{swp}(sw) * w) + \frac{1}{nb_{swr}} \times \sum_{r=1}^{nb_{swr}} (Cf_{swr}(sw)(1-w)). \qquad (4.1)$$

Après avoir établi le degré de confiance global placé dans un service Web, nous pouvons calculer la pertinence de chaque service dans le groupe. Comme mentionné précédemment la pertinence se calcule en fonction de la valeur de confiance $\mathrm{Cf}_{sw_{ij}}(sw)$ j=1..n, la valeur de réputation donnée par un client c $\mathrm{Rp}_c(sw)$ et la charge Q des requêtes exécutées par le service Web. La pertinence $\mathrm{Pert}_{GD_i}(sw_{ij})$ d'un service Web sw_{ij} au sein d'un groupe de diversité GD_i est donnée comme suit :

$$Pert_{GD}(sw) = Cf_{Cf_{sw_{ij}}(sw)}(sw) \times w_1 + Rp_c(sw) \times w_2 + Q_{sw} \times w_3. \qquad (4.2)$$

Avec w_1, w_2, w_3 représentent les poids donnés par un client aux différents paramètres de confiance, réputation et charge de travail.

4.3.4.2 Qualité d'un groupe de diversité

La qualité d'un groupe de diversité (*QoG*) est souvent définie en termes de disponibilité et de fiabilité de ces services Web membres. Elle est essentiellement fonction de la configuration du groupe, c.-à-d. , la sélection des membres et la manière dont ils interagissent (e.g., vote ou pas). Dans le cas parallèle, ces deux paramètres contrôlent le nombre acceptable de services Web pouvant être simultanément en échec au sein du groupe tout en continuant de fournir le service attendu. Par exemple, un groupe de diversité peut masquer les défaillances affectant la disponibilité de la minorité de ses services Web.

Comme nous le verrons plus tard, l'utilisation de services Web peu fiables peut donner lieu à un groupe de diversité tolérant aux fautes mais avec une faible fiabilité. Aussi, des services Web présentant une faible disponibilité peuvent conduire à la défaillance du groupe quand le nombre de redondance devient insuffisant pour masquer les défaillances [ZD05].

La détermination de la *QoG* permet de prendre une décision concernant les groupes formés précédemment. Cette décision concerne le redimensionnement et le changement des membres dans le groupe de diversité. En réalité, les membres d'un groupe de diversité peuvent changer même si la taille du groupe reste fixe. Ce cas de figure survient quand la *QoG* est sous un certain seuil (Cas d'ajout des services Web) suivant le type d'application $TA \in \{C, SC, NC\}$. Aussi quand la *QoG* est bonne et le type d'application est *NC*, des services Web doivent quitter le groupe pour éviter la consommation inutile des ressources.

Un groupe de diversité implémentant une fonctionnalité critique est plus exigeant en termes de *QoG* qu'un autre réalisant une fonctionnalité semi-critique ou non critique. La valeur de la qualité correspond à l'une des trois classes suivantes : pauvre, moyenne, ou bonne et ce en termes de pertinence des services Web dans ce groupe et de leur fiabilité. Une qualité de groupe (*QoG*) appartient à l'intervalle [seuil, Max] où le seuil défini par le concepteur ($seuil > 0$) selon le type d'application et Max représente la *QoG* maximale d'une classe particulière.

La qualité *QoG* d'un groupe GD_i avec une taille de n est donnée comme suit :

$$QoG = \frac{1}{n} \times \sum_{j=1}^{n} (Disp_{sw_{ij}}, Fiab_{sw_{ij}}, Pert_{GD_i}(sw_{ij}) \tag{4.3}$$

Sachant que $Disp_{sw_{ij}}$, $Fiab_{sw_{ij}}$ représentent respectivement la disponibilité et la fiabilité du service Web sw_{ij} et n est la taille du groupe.

4.3.4.3 Exigences d'un client

La stratégie de sélection adaptative a pour but de choisir la meilleure combinaison de services Web garantissant la meilleure qualité du groupe (*QoG*). Cette stratégie prend

en compte des contraintes de tolérance aux fautes exprimées par le client sous forme d'exigences assignées pour l'ensemble des fonctionnalités. Dans le cas d'une application temps réel, il s'agit de sélectionner la combinaison des services Web offrant la meilleure (QoG) en termes de contraintes temps réel et d'autres contraintes par rapport au groupe. Soit CR={f, t, r} l'ensemble des critères de sélection où f, t et r représentent respectivement, le nombre de fautes détectées dans le service Web, la durée d'exécution d'un service Web, et le coût d'exécution d'un service Web en termes de ressources utilisées (BDD, ressources calculatoires) ou des services Web payants pour compléter la tâche du service.

Les exigences des clients mesurent le taux toléré pour chaque critère dans CR. Par exemple un client exige que :

- La durée d'exécution d'un service Web ne doit pas dépasser un certain timing. Il représente ce critère (le taux d'exécution maximum toléré) pour le service Web en termes de t^{act}_{max} ($0 \leq t^{act}_{max} \leq 1$).
- Le taux de défaillance maximum toléré ne doit pas dépasser un seuil selon le type de la fonctionnalité demandée par les clients. Ce taux de défaillance est exprimé en termes de f^{act}_{max} ($0 \leq f^{act}_{max} \leq 1$).
- Le taux consommation maximum toléré des ressources est exprimé en termes de r^{act}_{max} ($0 \leq r^{act}_{max} \leq 1$). Cette contrainte ne doit pas dépasser le seuil accepté par l'application.

Par la suite, CR^{act}_{max} représente la liste des critères tolérés par un service Web, avec $CR^{act}_{max} = \{f^{act}_{max}, t^{act}_{max}, r^{act}_{max}\}$

4.3.5 Formalisation

Soit un groupe de diversité $GD_i = \{sw_{i1}, sw_{i2}, .. , sw_{im}\}$ avec m le nombre de services Web dans le groupe, soit l'historique des exécutions $log = \{\mathbb{T}, \mathbb{S}, \mathbb{R}, Cf_{sw_{ij}}(sw), CR^{his}_{max}, \mathcal{ORD}\}$, où :

- $\mathbb{T}$ correspond à l'ensemble des intervalles de temps d'éxécution de chaque service dans le log.
- $\mathbb{S}$ correspond à l'ensemble de tous les états de terminaison d'un service dans $\mathbb{T}$. Un état de terminaison peut prendre les valeurs suivantes : {Succès, Valeur, Crash}.
- $\mathbb{R}$ représente l'ensemble des coûts des ressources utilisées pour l'invocation des services dans un groupe GD.
- $Cf_{sw_{ij}}(sw)$ représente le degré de confiance des services Web membres sw_{ij} dà un service Web sw au sein du même groupe.
- CR^{his}_{max} : Le taux des critères maximum toléré dans les exécutions précédentes (his) pour les services exécutés dans un intervalle de temps.
- $\mathcal{ORD}$ représente la fonction d'ordonnancement des services dans un groupe invoqué séquentiellement (voir Section 4.4.4)

Le problème consiste à déterminer le nouveau Groupe GD′ assurant la fiabilité et la disponibilité du groupe conçu dans le cas d'un changement de contexte. La détermination du nouveau groupe GD_k={sw_{k1}, sw_{k2},..., sw_{kn}} revient à définir :

- La taille n du nouveau groupe (ou le degré de redondance)
- sw_{kj} à impliquer dans le nouveau groupe, j=1..n.

4.4 Calcul du degré de redondance dans le groupe de diversité

4.4.1 Hypothèses sur le nombre de fautes initialement tolérées

Dans un contexte de services Web, le protocole parallèle nécessite 2×f+1 répliques de service qui doivent exécuter la requête. Cela garantit qu'au moins $f+1$ services non défaillants vont l'exécuter. Traditionnellement, la procédure de vote attend de recevoir la réponse de 2×f+1 services et rend la réponse majoritaire de $f+1$ services. Néanmoins, face à des défaillances de type Byzantine et des défaillances de type crash qui se sont produites simultanément, 2×f+f′+1 services équivalents sont nécessaires pour tolérer f fautes Byzantine et f' fautes crash [RCS01].

Dans le protocole séquentiel, en présence d'une faute dans un service, un autre service est sélectionné pour recouvrir la défaillance causée. Donc $f+1$ services sont nécessaires pour tolérer f fautes. Dans le cas des fautes de type Byzantine une spécification sur le résultat attendu est nécessaire pour valider la réponse d'un seul service. En cas où le service Web crash ou la réponse n'est pas correcte (en comparant avec la spécification), un autre service équivalent traite la requête.

Le tableau 4.2 récapitule selon le mode (protocole d'invocation) et le types de fautes, le nombre de répliques nécessaires pour tolérer f fautes.

Table 4.2 – Nombre de services selon f fautes initialement tolérées

Mode / Type de fautes	Parallèle	Séquentiel
Crash	f+1	f+1
Byzantine	2× f+ 1	f+1 (Avec une spécification)

4.4.2 Principes

Étant donné un ensemble de services Web divers disponibles, un redimensionnement permettra d'ajuster automatiquement le degré de redondance noté n à utiliser pour s'accorder à l'évolution des fautes observées. En l'absence de telles fautes, le groupe peut réduire le nombre de services Web à impliquer pour éviter une dépense inutile de ressources. Lorsque le degré prévu de redondance n'est plus suffisant pour compenser

les perturbations actuelles, il serait avantageux de dynamiquement réviser ce degré et engager des services Web additionnels s'ils sont disponibles.

Une approche adaptative de tolérance aux fautes doit supporter le changement dans le nombre de fautes à tolérer. Ce changement peut prendre place si l'application augmente ou diminue le nombre de fautes détectées par rapport au nombre initial de fautes largement tolérées. Donc, il s'agit d'abord de déterminer le degré de redondance nécessaire pour répondre aux besoins en tolérance aux fautes de l'application cliente (ou utilisateur final) selon le protocole utilisé. L'algorithme 3 défini dans la section 4.4.3 (page 75) donne les différentes étapes pour déterminer à quel moment faut-il augmenter ou diminuer n selon le mode d'exécution (*Mode*) et le nombre de fautes initialement toléré (f_t). Par la suite, nous détaillons le principe utilisé pour chacun des deux protocoles.

Un travail intéressant est celui de Buys et al [BFB11] pour définir la pertinence d'un service Web au sein du groupe de diversité dans une procédure de vote. L'objectif est de s'assurer que le vote se termine dans les meilleures circonstances. Les auteurs proposent deux étapes :

- **Capturer la qualité de la configuration courante du groupe de diversité**. Pour cela, ils utilisent la métrique distance à la défaillance (en anglais, distance to failure ($dtof$)), introduite par De Florio [Flo09]. La $dtof$ est utilisée pour déterminer si la configuration actuelle du groupe est en mesure d'assurer sa disponibilité.
- **Quantifier l'impact d'un service Web sur l'efficacité du groupe de diversité dans le temps**. La mesure $dtof$ [DF09] permet de capturer l'impact instantané d'une configuration donnée sur l'efficacité du groupe de diversité mais n'est pas en mesure de capturer cet impact dans le temps. Un service Web échouant de manière répétée à fournir une contribution utile, la procédure de vote lui associera une dissidence normalisée élevée sous forme de pénalités.

4.4.3 Cas du Protocole parallèle

Dans le protocole parallèle, n services Web sont activés en même temps pour une exécution. Nous proposons de généraliser le concept de distance à la défaillance (dof) proposé par s[Flo09] utilisé pour mesurer les turbulences courantes de l'environnement dans le cas du vote.

L'équation 4.4 calcule dof où n est le degré de redondance et f_d est le nombre de votes différents de la majorité Maj si elle existe c.-à-d. le nombre de fautes détectées, sinon dof retourne 0. L'intuition derrière dof est de représenter de combien sommes nous proches de la défaillance après le dernier tour de vote dans le cas parallèle.

$$dof(n, f_d) = \lceil n/2 \rceil - f_d, \quad si \quad Mode = parallele \tag{4.4}$$

La décision concernant le redimensionnement du groupe dépend de trois cas de figures selon dof :

1. dof **= 0, Pas de changement de taille**. La configuration courante du groupe répond parfaitement aux besoins de tolérance aux fautes, en termes du nombre de répliques participant dans le groupe en question. Cela veut dire que le nombre de fautes f_t prévu es et le nombre de fautes f_d détectées sont égaux, par conséquent pas de changement dans la taille du groupe.

2. dof **< 0, Augmentation de taille**. La configuration courante peut causer une défaillance du groupe de diversité et par la suite de la composition en entier. Si le nombre de fautes augmente de f_t à f_d, $2 \times (f_d\text{-}f_t)$ nouvelle répliques sont nécessaires pour s'adapter à ce changement. Selon le type d'application considéré (C, SC, NC) et le nombre de fautes détectées, nous pouvons ajouter un **Min** de services égal à 2 si une faute additionnelle non prévue est détectée ou ajouter le **Max** correspondant à $n+1$ répliques si toutes les répliques sont défaillantes.

3. dof **> 0, Diminution de taille**. La taille du groupe c.-à-d. la configuration courante répond largement en termes du nombre de répliques aux besoins en tolérance aux fautes. Pour garantir une utilisation efficace des ressources, la taille du groupe doit être réduite. Si le nombre de fautes initialement tolérées est f_t et que l'application détecte f_d fautes sachant que $f_d < f_t$, $2 \times (f_t\text{-}f_d)$ répliques doivent quitter le groupe.

La figure 4.2 montre un exemple avec trois répliques sémantiquement équivalentes (Taille initial du GD_1 =3). Au moment du vote, si la majorité $Maj = 2$ approuve la même valeur, $dof = \lceil n/2 \rceil - f_d$ =0, le résultat sera renvoyé au demandeur du service comme vu dans (A). Dans ce cas de figure, la taille du groupe ne change pas car la configuration actuelle répond aux besoins en TF.

Sinon si aucune majorité n'est approuvée (dof<0), le nombre de fautes probables dans le groupe peut être 2 (cas du (B)) comme il peut être 3 (tous les services sont défaillants) (cas du (C)). Pour tolérer cette augmentation dans le nombre de fautes il faut prévenir un minimum de répliques ($Min = 2$) c.-à-d. 2×(nouveau nombre de fautes - le précédent nombre de fautes) =(2×(2-1)=2 répliques), ou toutes les répliques sont défaillantes c.-à-d. , le f_d=3. Par la suite il faut ajouter 4 répliques correspondant à (2×(3-1)=4. La nouvelle taille du groupe sera $n = 7$ pour tolérer 3 fautes. Dans ce cas la majorité sera atteinte si $Maj = 3$ approuve le même résultat. De manière similaire si $dof > 0$, la taille du groupe sera diminué en fonction du nombre de fautes détectées.

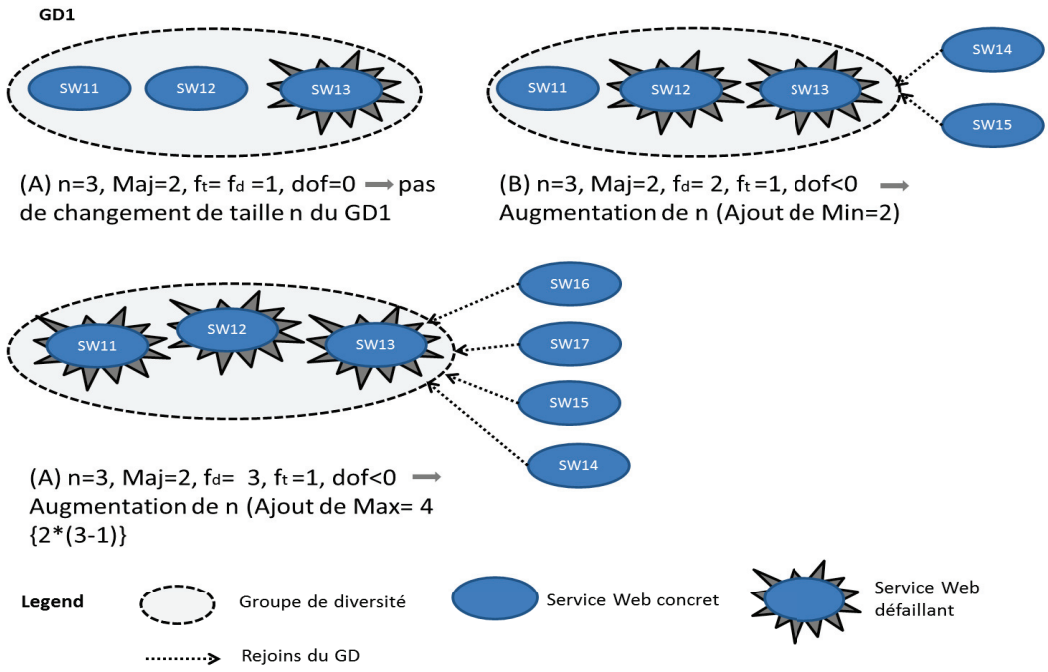


FIGURE 4.2 – **Adaptation au changement du nombre de fautes**

4.4.3.1 Algorithme

L'algorithme 3 calcule le degré de redondance exigé par une application dans le protocole parallèle. Il prend comme paramètres : le protocole d'activation des services Web *Protocole*, la criticité de la fonctionnalité noté *TA*, le nombre de fautes initiales f_t et la pertinence du service Web défaillant dans le groupe $\text{Pert}_{GD_i}(\text{sw}_{Ech})$ et retourne le degré de redondance n.

Dans la première partie, il s'agit de calculer la taille initiale du groupe de diversité n, dans le cas du protocole parallèle $2\times f_t+1$ répliques sont nécessaires pour former le groupe de diversité qui tolère f_t fautes Byzantine (i.e. en valeur), et f_t+1 répliques tolère f_t fautes de type Crash dans le protocole séquentiel. Cependant, dans des applications possédant une spécification sur le résultat retourné, un groupe de f_t+1 répliques peut tolérer f_t fautes en valeur. Nous considérons dans cet algorithme des applications sans spécification, donc pour tolérer f_t fautes en valeur il faut $2\times f_t+1$ répliques.

Dans la deuxième partie, il s'agit de calculer la distance de défaillance dof (Équation 4.4) pour décider de la reconfiguration ou non. Par la suite, la taille du groupe de diversité est adaptée au changement de nombre de fautes détectées. Le tableau 4.3 récapitule selon le type d'application TA={*C*, *SC*, *NC*} les différentes combinaisons où il faut augmenter ou diminuer le nombre de services Web avec le $Max= 2 \times (f_d\text{-}f_t)$ ou le

Min=2. La pertinence est classée selon trois valeurs : Pauvre, Moyenne ou Élevé. Cette classification dépend du type d'application c.-à-d. pour la même valeur de pertinence, un service Web peut appartenir à la classe "Moyenne" dans un groupe critique et "Élevé" pour un groupe semi critique.
Le nombre de services Web à impliquer dans le groupe de diversité est déterminé par rapport à la criticité de la fonctionnalité à implémenter par le groupe. Les groupes *C* et *SC* exigent un résultat correct et cohérent.

Un service Web avec une valeur de pertinence élevée dans un groupe *C* doit être remplacé de telle sorte à garantir le degré de redondance qui correspond au *Max* de services Web et par la suite le bon fonctionnement du groupe. Du même, pour un groupe *SC*, les services Web avec une valeur de pertinence élevée ou moyenne doivent être remplacés de telle sorte à garantir le *Max*, car il une fonctionnalité *SC* tolère les réponses incomplètes mais non pas les réponses incorrectes, donc la prévision du *Max* de services est nécessaire. Par contre, si la pertinence d'un service Web est pauvre ce dernier doit être remplacé de telle sorte à garantir le *Min* de services dans le groupe afin d'optimiser la consommation des ressources.

Dans le cas d'un groupe à fonctionnalité *NC*, le manager du groupe peut ignorer le résultat incorrect d'un service Web pertinent, donc l'ajout d'un *Min* sera suffisant pour remédier aux fautes détectées sinon, le résultat du service Web pertinent sera ignoré.

TABLE 4.3 – Nombre de répliques selon le *TA* et pertinence du services Web dans GD

Pertinence / TA	Pauvre	Moyenne	Élevé
C	$2 \times (f_d\text{-}f_t)$	$2 \times (f_d\text{-}f_t)$	$2 \times (f_d\text{-}f_t)$
SC	2	$2 \times (f_d\text{-}f_t)$	$2 \times (f_d\text{-}f_t)$
NC	2	2	2

Algorithme 3: Calcul de degré de redondance et sa variation

Données : *Protocole,* f_t, $Pert_{GD_i}(sw_{Ech})$, *TA*

Résultat : *n : Nombre de services à impliquer dans le groupe*

```
si (Protocole =Parallel) alors
    n=2× f_t+1                          % Nombre de services initialement définis%
Calculer dof selon Équation 4.4
suivant dof faire
    case dof<0                          % Augmentation de taille%
        suivant TA et Pert_GDi(sw_Ech) faire
            case TA = C
                n = n+ 2 ×(f_d-f_t)     % Type d'application critique%
            case TA = SC
                si (Pert_GDi(sw_Ech)="Pauvre") alors
                    n=n+2
                sinon
                    n = n+ 2 ×(f_d-f_t)
            case TA = NC
                n = n+ 2
    case dof>0                          % Diminution de taille%
        si n<3 alors
            n=3
        suivant TA et Pert_GDi(sw_Ech) faire
            case TA = C
                n -2                    % Supprimer le Min de services%
            case TA = SC
                si (Pert_GDi(sw_Ech)="Pauvre") alors
                    n=n-2×(f_t-f_d)
                sinon
                    n = n-2 ×(f_t-f_d)
            case TA = NC
                n = n- 2×(f_t-f_d)
    case dof=0
        % Pas de changement de taille, si QoG < Seuil, changement du membres%
Retourner n
```

4.4.4 Cas du Protocole séquentiel : Réservation et ordonnancement des services

Dans une stratégie séquentielle, il s'agit de réserver les services Web les plus fiables et de définir leur ordonnancement.

4.4.4.1 Réservation des services Web :

La réservation est faite comme dans le protocole parallèle en fonction des exigences clients, du type d'application (C, SC, NC) et du type de défaillance à tolérer (Byzantine, crash). Dans le protocole séquentiel un seul service est invoqué à la fois. Lorsque ce service tombe en panne, un autre service sera sélectionné pour le remplacer. Donc le nombre de services Web à ajouter dans un groupe est toujours 1. En réalité contrairement au protocole parallèle, des défaillances simultanées dans les services Web au sein du groupe de diversité ne peuvent pas se produire dans le protocole séquentiel c.-à-d. un seul service est défaillant à la fois. Par contre, pour une invocation d'un groupe de diversité, plusieurs exécutions des services Web peuvent être envisagées pour répondre à une demande. En outre, le manager du groupe de diversité invoque les services Web l'un après l'autre jusqu'à l'obtention d'un résultat ou l'épuisement des services Web disponibles. Cela signifie que réellement le nombre de fautes détectées pour le groupe de diversité $n-1$ si le groupe contient n services Web. Le but de réservation est de définir la taille du groupe au préalable et de l'adapter au changement du contexte en termes du nombre de fautes détectées. L'avantage de réserver les n services pour un groupe de diversité est d'éviter dans des fonctionnalités critiques que les services Web ne soient pas disponibles (par exemple dans le cas où un autre groupe accède aux services Web disponibles pour la même fonctionnalité).

Nous calculons comme dans le protocole parallèle, la distance à la défaillance (dof) pour prendre une décision concernant le groupe courant. L'équation 4.5 calcule dof où f_d est le nombre de fautes détectées dans un groupe de diversité c.-à-d. si on a un groupe de diversité GD= $\{sw_1, sw_2, sw_3\}$ avec cet ordonnancement où sw_3 a été exécuté, le nombre de fautes détectées sera f_d=2. f_t représente le nombre de fautes tolérées qui est 1 dans un groupe avec la stratégie séquentielle.

$$dof(n, f_d) = f_t - f_d, \quad si \quad Mode = Sequentiel \tag{4.5}$$

Comme dans la stratégie parallèle, la décision concernant le redimensionnement du groupe dépend de trois cas de figure selon dof. Nous parlons en ce qui suit de la confi-

guration du groupe, où le groupe contient tous les services Web exécutés pour la fonctionnalité qu'il implémente, et nous parlons de taille du groupe en termes du nombre de services Web à réserver :

1. dof **= 0, Pas de changement de taille**. $f_d = f_t$ La configuration courante du groupe répond parfaitement au besoin de tolérance aux fautes.
2. dof **< 0, Augmentation de taille**. La configuration courante peut causer une défaillance du groupe de diversité. Si le nombre de fautes augmente de f_t à f_d, (f_d-f_t) nouvelles répliques sont nécessaires pour s'adapter à ce changement. Selon le type d'application considéré (C, SC, NC) et le nombre de fautes détectées, nous pouvons ajouter un **Min** de services égal à 1 si une faute additionnelle non prévue est détectée ou ajouter le **Max** correspondant à 2 répliques si toutes les répliques sont défaillantes. La réservation de Max dans des groupes qui implémentent une fonctionnalité critique correspond à 2 services Web au plus (n=3, le nombre des services Web dans le groupe). Dans le cas d'un groupe de diversité qui implémente une fonctionnalité non critique où la défaillance peut être ignorée, la réservation de $Min = 1$ est suffisante pour un GD (n=1).
3. dof **> 0, Diminution de taille**. La configuration courante répond largement en termes du nombre de répliques aux besoins en TF. Si le nombre de fautes initialement tolérées est f_t et que l'application détecte f_d fautes sachant que $f_d < f_t$, (f_t-f_d) répliques doivent quitter le groupe.

Après avoir déterminé le degré de redondance nécessaire, la détermination des services Web membres à impliquer dans le groupe de diversité est faite grâce à la fonction de sélection calculée dans la section 4.6. Contrairement au protocole parallèle où tous les services Web sélectionnés sont invoqués, dans le protocole séquentiel, un seul service Web parmi les services Web réservés doit jouer le rôle du primaire. La détermination du service web primaire et des services web secondaires revient à déterminer l'ordonnancement de ces derniers. Imaginons un groupe de diversité GD_i ={sw_{i1}, sw_{i2}, sw_{i3}} formé dans le passé avec cet ordonnancement d'exécution. Cela implique que sw_{i1} représentait le service primaire, qu'il a été invoqué et qu'il a échoué, ensuite le service secondaire sw_{i2} remplace sw_{i1} et devient primaire mais lui aussi a rendu un résultat inattendu, le service sw_{i3} remplace sw_{i2} et devient primaire, et rend un résultat correct. L'objectif derrière la détermination de l'ordonnancement est de placer le service Web sw_{i3} comme étant le primaire dès le début, s'il remplace fréquemment avec succès les autres services dans GD_i. Nous choisissons d'utiliser la notion d'itemset fréquent pour détecter la fréquence d'un service Web dans les exécutions précédentes. En réalité, le problème

d'ordonnancement se résume en la détection des itemsets fréquents collectés dans le fichier Log, ce qui garantit que le groupe de diversité actuel rend un résultat correct et cela dans les meilleurs délais. La section suivante (Section 4.4.4.2) explique les différentes notions de base pour définir l'ordonnancement des services Web au sein d'un groupe de diversité invoqué séquentiellement.

4.4.4.2 Itemset fréquent et motif séquentiel fréquent :

Pour faire face aux besoins des nouvelles applications (médicales, suivi de consommation, suivi des navigations sur un serveur Internet, etc), de plus en plus de données sont stockées sous la forme de séquences. La recherche des régularités dans les bases de données est l'idée principale. Ces régularités s'expriment sous différentes formes. Dans l'analyse du panier d'achats de consommateurs, l'extraction des **itemsets** consiste à mettre en exergue les cooccurrences entres les produits achetés c.-à-d. déterminer les produits (les items) qui sont " souvent " achetés simultanément. On parle alors d'itemsets fréquents. Par exemple, en analysant les tickets de caisse d'un supermarché, on pourrait produire des itemsets (un ensemble d'items) de type " le pain et le lait sont présents dans 10% des caddies ". Un motif fréquent consiste à un ensemble d'itemsets, avec une fréquence non inférieure à un seuil défini par l'utilisateur. Par exemple, l'achat d'un PC en premier, suivi d'un achat d'un appareil photo numérique, puis une carte mémoire, si elle se produit fréquemment dans une base de données de l'historique des achats, la sous séquence représente un motif séquentiel fréquent.

Extraction des motifs séquentiels et des itemset fréquents : L'extraction des motifs séquentiels fréquents à été introduit par Agrawal and Srikant [AS95], est devenue un problème important dans l'extraction de données. Une séquence dans une base de données consiste à des éléments ou des évènements ordonnés, enregistrés avec ou sans une notion concrète de temps. La recherche des itemsets fréquents est souvent présentée comme un préalable à l'extraction des règles d'association [GBYMN07] où l'on essaie, de mettre en évidence des relations de causalité. Par exemple, une règle possible serait " ceux qui ont acheté du pain et du lait ont aussi acheté du beurre ". L'objectif est d'exploiter ce type de connaissance pour mieux agencer les rayons (mettre le beurre pas trop loin du pain et du lait) ou pour faire une offre promotionnelle ciblée (faire une promotion sur le pain et le lait dans le but d'augmenter les ventes de beurre).

En réalité, les itemsets fréquents sont en eux-mêmes porteurs d'informations. Savoir quels sont les produits achetés ensembles permet d'identifier les liens existants entre eux et, par là, de réaliser une typologie des achats ou de dégager des comportements

types chez les consommateurs.

Plusieurs algorithmes de recherche d'extraction de motifs séquentiels ont été proposés dans la littérature. De manière chronologique les premières méthodes proposées pour résoudre ce problème d'extraction étaient basées sur le principe d'une recherche "en largeur d'abord" (`breadth-first`) qui correspond au modèle d'Apriori [AMS+96] à savoir GSP [SA96], qui est l'algorithme pionnier de la recherche de motifs séquentiels généralisés ainsi que SPADE [Zak01]. La communauté s'est ensuite penchée sur le développement de techniques dites "en profondeur d'abord" (`depth-first`) très bien adaptées à cette problématique. Les algorithmes implémentant ce principe sont par exemple PSP [MCP98], PREFIXSPAN [HPMA+00] ou encore SPAM [AFGY02] qui est le premier à extraire des motifs séquentiels à l'aide de bitmaps. D'autres méthodes ont été proposées pour la recherche incrémentale de motifs séquentiels, nous citons la méthode ISE [MPT03] et FASTUP [LL04].

Nous introduisons en ce qui suit les concepts de séquence d'itemset, les itemsets fréquents et nous expliquons comment ces informations peuvent nous être utiles pour la détermination de l'ordonnancement des services Web dans la stratégie séquentielle.

4.4.4.3 Processus d'ordonnancement

Le but de cette section est d'expliquer à travers les différentes définitions, le processus d'ordonnancement.

Définition 4.1. *(**Séquence d'Itemset**.) Une transaction constitue, pour un client C, l'ensemble des items achetés par C à une même date. Dans une base de données client, une transaction s'écrit sous la forme d'un ensemble : id-client, id-date, itemset. Un itemset est un ensemble d'items non vide noté (i_1,i_2,...,i_k) où i_j, avec j de 1 à k. Une séquence est une liste ordonnée, non vide, d'itemsets notée < s_1, s_2,..., s_n > où s_j est un itemset (une séquence est donc une suite de transactions qui apporte une relation d'ordre entre les transactions).*

Soit C un client et S = < (C) (D E) (H) >, la séquence de données représentant les achats de ce client. S peut être interprétée par "C a acheté l'item C, puis en même temps les items D et E et enfin l'item H".

Dans notre cas une séquence de donnée représente une exécution d'une séquence de services Web équivalents. Le problème d'ordonnancement des services Web pour une invocation séquentielle revient donc à détecter les différents itemsets fréquents dans un fichier Log contenant les services Web invoqués dans le passé. En réalité dans une séquence d'itemset I=<sw_1, sw_2, sw_3> si le service Web sw_3 a été invoqué cela implique

que les deux services sw_1 et sw_2 ont échoué. Par la suite, l'invocation du service sw_3 en premier peut garantir un résultat correct dans les meilleurs délais.

Soit I={sw_1, sw_2,... sw_m} un itemset contenant l'ensemble des items (services), représentant une séquence d'exécution des services Web dans le fichier Log. Soit un vecteur T= (t_1, t_2, t_n) une transaction sur I. Soit $\mathbb{SIP}_{min}$ un nombre représentant le support minimum désiré (seuil). Nous cherchons à définir l'ordonnancement des n meilleurs services Web sélectionnés.

Définition 4.2. ***(Support d'itemset).*** *Le support $\mathbb{SIP}(sw_i)$ d'un itemset à 1 item (sw_i) est égal au nombre de transactions dans lesquelles l'item (sw_i) apparaît. Le support d'un itemset de 2 items $\mathbb{SIP}(sw_i, sw_j)$ représente le nombre de transactions dans lesquelles l'ensemble ordonné (sw_i, sw_j) apparaît.*

Définition 4.3. ***(Itemset fréquent).*** *Un itemset I est dit fréquent si son support est supérieur au seuil pré-défini $\mathbb{SIP}_{min}$. Nous notons I_f l'ensemble d'itemset fréquent tel que :*

$$I_f = \{\ I' \subset I \mid \mathbb{SIP}(I') \geq \mathbb{SIP}_{min}\ \}.$$

La définition suivante permet de déterminer l'ensemble des services Web qui ont remplacé un service sw_i.

Définition 4.4. ***(Remplaçants d'un service).*** *Soit $\mathcal{L}$ l'ensemble des services Web équivalents disponibles. Nous définissons $\mathcal{RP}_{sw_i}$ l'ensemble des items (Services) qui ont remplacer sw_i comme suit : $\mathcal{RP}_{sw_i} = \{\ sw_k \in \mathcal{L} \mid (sw_i, sw_k) \subset I_f\ \}$*

La fonction suivante permet le calcul de la cardinalité des différents sous ensembles contenant un item (Service) particulier. Mathématiquement, il s'agit d'identifier à partir d'un ensemble $\mathbb{X}$, les différents sous ensembles contenant un élément e, et par la suite calculer la fréquence d'apparition du sous ensemble contenant e dans l'ensemble $\mathbb{X}$. Cela va nous permettre de calculer pour chaque service Web dans le groupe sélectionné, la fréquence d'apparition du service dans le fichier Log, mais aussi d'identifier si l'apparition consistait en un remplacement d'un autre service ou un remplacement par un autre service.

Définition 4.5. ***(Fonction d'apparition).*** *Étant donné un Log, nous appelons $F^{+}_{sw_i}$ la fonction qui calcul le nombre de fois où sw_i à remplacer sw_k. Nous définissons $F^{+}_{sw_i}$ comme suit $F^{+}_{sw_i} = Card\ \{\ \{\ sw_k, sw_i\} \mid sw_i \in \mathcal{RP}_{sw_k}\}$. Pareil nous appelons $F^{-}_{sw_i}$ la fonction qui calcul le nombre de fois où sw_i a été remplacé par sw_k, $F^{-}_{sw_i} = Card\ \{\ \{\ sw_i, sw_k\} \mid sw_k \in \mathcal{RP}_{sw_i}\}$.*

Après avoir identifié la fonction d'apparition pour chaque service dans le groupe actuel, nous pouvons définir le support positif et négatif d'un service Web.

Définition 4.6. ***(Support positif et Support négatif).*** *Nous définissons* $\mathbb{SIP}^{+}_{sw_i}$ *le support positif du service Web* SW_i *comme suit :* $\mathbb{SIP}^{+}_{sw_i} = F^{+}_{sw_i} - F^{-}_{sw_i}$ *tel que* $F^{+}_{sw_i} > F^{-}_{sw_i}$. *Du même, le support négatif* $\mathbb{SIP}^{-}_{sw_i}$ *du service Web* sw_i *est défini comme étant :* $\mathbb{SIP}^{-}_{sw_i} = F^{+}_{sw_i} - F^{-}_{sw_i}$ *tel que* $F^{+}_{sw_i} < F^{-}_{sw_i}$

Nous définissons maintenant les différents cas possibles d'un ordonnancement. Nous partons d'un ensemble des meilleurs services Web sélectionnés, le but est de placer les services Web dans un ordre d'exécution qui garantit une meilleure qualité de groupe en termes de temps et surtout de bon fonctionnement.

Définition 4.7. ***(Fonction d'ordonnancement).*** *Soit* $\mathcal{ES}^{+}$ *l'ensemble de services Web remplaçant, et* $\mathcal{ES}^{-}$ *la liste des services Web qui doivent être remplacés. Soit* DG^{act} *le groupe de diversité actuel qui contient les meilleurs services Web. Nous définissons* $\mathcal{ORD}_{sw_i,sw_j}$ *la fonction qui place* sw_i *avant* sw_j *comme suit :*

- $\forall\ sw_k \in DG^{act}$ | *Si* $\exists\ (sw_k, sw_i) \subset I_f$ *et* $\exists\ (sw_k, sw_j) \subset I_f$ *tel que* $\mathbb{SIP}^{+}_{sw_i} = \mathbb{SIP}^{+}_{sw_j}$ *alors :*
  1. *Si* $F^{+}_{sw_i} > F^{+}_{sw_j}$ *et* $\mathbb{SIP}(sw_i, sw_j) > \mathbb{SIP}(sw_j, sw_i)$ | *alors* $\mathcal{ORD}_{sw_j,sw_i}$.
  2. *Si* $F^{+}_{sw_i} > F^{+}_{sw_j}$ *et* $\mathbb{SIP}(sw_i, sw_j) < \mathbb{SIP}(sw_j, sw_i)$ | *alors* $\mathcal{ORD}_{sw_i,sw_j}$.
  3. $\mathcal{ORD}_{sw_i,sw_j}$ *ou* $\mathcal{ORD}_{sw_j,sw_i}$, *sinon*
- $\forall\ sw_k \in DG^{act}$ | *Si* $\exists\ (sw_k, sw_i) \subset I_f$ *et* $\exists\ (sw_k, sw_j) \subset I_f$ *tel que* $\mathbb{SIP}(sw_k, sw_i) = \mathbb{SIP}(sw_k, sw_j)$ *et* $\mathbb{SIP}^{+}_{sw_j} > \mathbb{SIP}^{+}_{sw_i}$ *alors* $\mathcal{ORD}_{sw_j,sw_i}$.
- $\forall\ sw_k \in DG^{act}$ | $\exists\ (sw_k, sw_i) \subset I_f$ *et* $\exists\ (sw_j, sw_k) \subset I_f$, *tel que* $\mathbb{SIP}^{-}_{sw_i} = \mathbb{SIP}^{-}_{sw_j}$ *et* $sw_k \in \mathcal{ES}^{-}$
- $\forall\ sw_k \in DG^{act}$ | *Si* $\exists\ sw_i$ *tel que* $sw_i \in \mathcal{RP}_{sw_k}$ *et* $sw_k \in \mathcal{ES}^{-}$ *alors* $\mathcal{ORD}_{sw_i,sw_k}$.

Exemple : Une transaction dans une base de données représente une séquence d'invocations de services Web. Soient les transactions suivantes :

Soit $\mathcal{L} = \{sw_1, sw_2, sw_3, sw_4, sw_5, sw_6\}$ l'ensemble de services Web équivalents disponibles. Soit un fichier Log contenant 10 observations (correspondant à des invocations de groupes de diversité incluant les services Web équivalents disponibles), et 6 items (services) :

< $\{sw_1, sw_3\}$, $\{sw_5, sw_2\}$, $\{sw_4, sw_1, sw_5, sw_3\}$,$\{sw_5, sw_2\}$, $\{sw_2, sw_3\}$, $\{sw_5, sw_6, sw_2\}$, $\{sw_4, sw_1, sw_2\}$, $\{sw_1, sw_2, sw_3\}$, $\{sw_1, \mathrm{sw}_5, sw_2\}$>

Le tableau suivant 4.4 résume les différents itemsets, le calcul des items fréquents repose sur la fréquence des items dans les itemsets suivants :

TABLE 4.4 – Liste des itemsets

0 items	1 item	2 items
$\emptyset$	$(sw_1) : 5$	$(sw_1, sw_2) : 3$
	$(sw_2) : 6$	$(sw_1, sw_3) : 3$
	$(sw_3) : 4$	$(sw_1, sw_5) : 2$
	$(sw_4) : 2$	$(sw_2, sw_3) : 2$
	$(sw_5) : 5$	$(sw_4, sw_1) : 2$
	$(sw_6) : 1$	$(sw_4, sw_2) : 1$
		$(sw_4, sw_3) : 1$
		$(sw_4, sw_5) : 1$
		$(sw_5, sw_2) : 4$
		$(sw_5, sw_3) : 1$
		$(sw_5, sw_6) : 1$
		$(sw_6, sw_2) : 1$

Pour une valeur de $\mathbb{SIP}_{min}$=2, nous avons 6 itemsets fréquents à 2 items. Nous considérons dans le calcul d'ordonnancement des services Web, les itemsets avec 2 items.

La liste des itemsets fréquents est I_f = {(sw_1, sw_2),(sw_1, sw_3), (sw_1, sw_5), (sw_2, sw_3), (sw_4, sw_1), (sw_5, sw_2)}.

La liste des remplaçant de chaque service dans les itemsets fréquents est comme suit : $\mathcal{RP}_{sw_1}$ = { $\{sw_k\} \in \mathcal{L}$ tel que : $\exists$ k | $(sw_1, sw_k) \subset I_f$ }

$\mathcal{RP}_{sw_1}$ = {sw_2, sw_3, sw_5}. Du même :

$\mathcal{RP}_{sw_2}$ = {sw_3}

$\mathcal{RP}_{sw_3}$ = $\emptyset$

$\mathcal{RP}_{sw_4}$ = {sw_1}.

$\mathcal{RP}_{sw_5}$ = {sw_2}.

La fonction d'apparition des services Web correspond à : $F^+_{sw_1}$ = { $\{sw_i, sw_1\} \in$ | $sw_1 \in \mathcal{RP}_{sw_i}$} $\Rightarrow$ $F^+_{sw_1}$ = 2

$F^-_{sw_1}$ = { $\{sw_1, sw_i\} \in$ | $sw_i \in \mathcal{RP}_{sw_1}$} $\Rightarrow$ $F^-_{sw_1}$ = 8

Cela implique que $\mathbb{SIP}^+_{sw_1}$ = 0 et $\mathbb{SIP}^-_{sw_1}$ = -6 et Pareil pour les autres services :

$F^+_{sw_2}$ = 7 $F^-_{sw_2}$ = 2 $\Rightarrow$ $\mathbb{SIP}^+_{sw_2}$ = 5 et $\mathbb{SIP}^-_{sw_1}$ = 0

$F^+_{sw_3}$ = 5 $F^-_{sw_3}$ = 0 $\Rightarrow$ $\mathbb{SIP}^+_{sw_3}$ = 5 et $\mathbb{SIP}^-_{sw_1}$ = 0

$F^+_{sw_4}$ = 0 $F^-_{sw_4}$ = 2 $\Rightarrow$ $\mathbb{SIP}^+_{sw_4}$ = 0 et $\mathbb{SIP}^-_{sw_1}$ = -2

$F^+_{sw_5}$ = 2 $F^-_{sw_5}$ = 4 $\Rightarrow$ $\mathbb{SIP}^+_{sw_5}$ = 0 et $\mathbb{SIP}^-_{sw_1}$ = -2

Supposons maintenant que la réservation de n services Web corresponds au 5 services : sw_1, sw_2, sw_3, sw_4, sw_5.

Nous commençons par ordonner le couple de services avec un support positif (sw_2, sw_3). Nous remarquons par ailleurs que les services sw_2, sw_3 ont le même support ($\mathbb{SIP}^{+}_{sw_3}$ = $\mathbb{SIP}^{+}_{sw_2}$). Nous avons $F^{+}_{sw_2} > F^{+}_{sw_3}$. Ceci veut dire que le service Web sw_2 a remplacé d'autres services dans le passé et que sw_3 a remplacé sw_2, ainsi $\mathbb{SIP}(sw_2, sw_3) > \mathbb{SIP}(sw_3, sw_2)$. La fonction d'ordonnancement est alors $\mathcal{ORD}_{sw_3,sw_2}$.

Maintenant, le processus d'ordonnancement ordonne le couple (sw_4, sw_5) qui possède la même valeur de support négatif $\mathbb{SIP}^{-}_{sw_4} = \mathbb{SIP}^{-}_{sw_5}$. Dans la liste des itemsets fréquents $\exists\ sw_1$ tel que $(sw_1, sw_5) \subset I_f$, $(sw_4, sw_1) \subset I_f$ et $sw_1 \in \mathcal{ES}^{-}$ c.-à-d. le service Web sw_1 à remplacer le service sw_4 et ce même service (sw_1) a été remplacé par le sw_5. Ceci implique que le processus d'ordonnancement doit placer sw_5 avant sw_4 pour éviter des invocations inutiles. La fonction d'ordonnancement sera $\mathcal{ORD}_{sw_5,sw_4}$.

Enfin malgré que $(sw_4, sw_1) \subset I_f$, cela n'empêche pas que sw_4 sera classé avant sw_1 car dans toutes les exécutions sw_1 a été remplacé et son support négatif $\mathbb{SIP}^{-}_{sw_1} < \mathbb{SIP}^{-}_{sw_4}$. La fonction d'ordonnancement entre sw_4 et sw_1 est $\mathcal{ORD}_{sw_4,sw_1}$

Après avoir déterminé le nombre de services Web à impliquer dans un *GD* invoqué séquentiellement, la prochaine étape consiste à identifier les services Web membres en utilisant la fonction de sélection définie dans la 4.6. Cette fonction prend en considération l'historique des services Web, le type d'application et les exigences clients actuelles en termes de tolérance aux faute et de *QoS*.

4.5 Historique d'éxecutions des services Web

La collecte de données est un processus de découverte de connaissances afin de comprendre l'activité d'un système et de diagnostiquer les problèmes. Elle se base sur un fichier Log contenant les évènements observés au moment de l'exécution. Dans notre travail, il s'agit de déterminer le comportement des services Web dans leurs précédentes exécutions en termes du nombre de fautes observées, le temps d'exécution des services Web, le coût, etc.

Nous identifions trois niveaux de collecte des données : **i)** Service Web (i.e., fiabilité et disponibilité) ; **ii)** Application cliente (e.g., besoins de l'utilisateur final ou environnement exécution du groupe de diversité) ; **iii)** Sous-groupes antérieurs (fiabilité et disponibilité).

4.5.1 Niveau Service Web

Les données collectées au niveau Service Web sont le temps de réponse et la charge (c.-à-d. le nombre de requêtes en attente). La contribution d'un service Web au succès du groupe peut dépendre d'autres aspects de son état opérationnel (c.-à-d. retour d'un

résultat correct ou pas). Certaines applications à caractère critique exigent un résultat dans un délai strict. Par exemple, si un groupe de diversité suit le protocole parallèle avec vote afin de retourner une réponse avant ce délai, le mécanisme de vote doit imposer un temps de réponse au delà duquel un service Web serait considéré comme défaillant et aurait un impact négatif sur la *QoG* de la configuration du groupe [BS].

Le groupe doit aussi être en mesure d'équilibrer la charge de manière optimale entre les services Web. En effet, le nombre de requêtes en cours de traitement par un service Web membre peut impacter la *QoG*.

Concrètement les données à extraire pour un service Web i sont :

1. Le taux de succès (S_i), le taux de défaillance f_{obs} d'un service Web (sw_i) avec le type l'application où le service a été exécuté. Nous cherchons avec ces données à déterminer si un service Web a contribué au succès ou a causé la défaillance d'un groupe critique *C*, Semi critique *SC* ou non critique *NC* ?
2. Le taux d'invocation d'un service Web en termes de délai de connexion à ce service et de temps de réponse (t_{obs}).
3. Le taux de consommation de ressources (r_{obs}) d'un service Web incluant le coût des ressources réservées pour ce service. Dans le cas d'invocation de services peu fiables, une réservation d'un grand nombre de ressources est nécessaire pour masquer les défaillances présentées dans le système.

4.5.2 Niveau Application Cliente

Nous identifions plusieurs types de données à collecter relatives aux besoins en *QoS* de l'application cliente (ou l'utilisateur) :

1. Le temps maximum de réponse (t_{max}^{his}) toléré par les clients de services dans les exécutions précédentes (e.g., sensible à la latence).
2. Le taux maximum de défaillance matérielle et/ou logique toléré dans les exécutions précédentes (f_{max}^{his}).
3. Le taux maximum toléré de consommation de ressources dans les exécutions précédentes (r_{max}^{his}).

4.5.3 Niveau Sous-Groupe

Les données collectées sont relatives à l'historique des interactions entre les services Web membres d'un groupe de diversité formé précédemment (e.g., sous-groupes avec une fiabilité élevée). Selon le degré de confiance entre deux fournisseurs (F_1, F_2) de services Web équivalents, la collaboration de leurs services respectifs sw_1 et sw_2 peut aboutir à un échec, alors que celle de sw_1 et sw_3 peut conduire à un succès. Si la confiance entre F_1 et F_2 est élevée, le groupe de diversité impliquant sw_1 et sw_3 fonctionne mieux que celui contenant sw_1 et sw_2. Aussi, deux services provenant d'un même fournisseur

peuvent approuver plus facilement le même résultat lors de l'utilisation d'un algorithme de consensus. Si F_1 ne place pas une confiance dans F_2, tout service de F_1 peut suspecter le résultat provenant des services de F_2.

Une autre information importante à extraire à partir du *Log* est liée à l'ordonnancement des services dans un protocole séquentiel. Par exemple un service sw_{ij} remplace **toujours** avec succès un service sw_{ik} (sw_{ij}, $sw_{ik} \in GD_i$), cette information doit apparaitre dans le *Log* comme étant un motif fréquent (e.g., [PYB+99]). La liste des itemsets **défini dans la section 4.4.4** est composée de différents ordonnancements d'exécution de services Web dans un protocole d'invocation séquentiel.

4.5.4 Monitoring du fichier Log

Le tableau 4.5 récapitule la classification des données à collecter à différents niveaux. Ces données sont enregistrées sous forme d'un tableau appelé tableau de Monitoring du fichier *Log* illustré dans 4.6.

TABLE 4.5 – Classification des données à collecter

Niveaux	Données à collecter	Descriptions
Service Web	t_{obs}	Temps de réponse
	f_{Valeur}	Évènement de défaillance Byzantine d'un service $f_{Valeur} \in f_{obs}$
	f_{Crash}	Évènement de défaillance d'un service par crash $f_{Crash} \in f_{obs}$
	r_{obs}	Ressources consommées
Application Cliente	t_{max}^{his}	Le temps maximum de réponse toléré dans les exécutions précédentes
	f_{max}^{his}	Le taux de défaillance largement toléré dans les exécutions précédentes
	r_{max}^{his}	Taux tolérées de consommation des ressources dans les exécutions précédentes
Sous-Groupe	$Cf_{sw_{ij}}(sw)$	Le degré de confiance des services Web sw_{ij} au sein du GD_i à un service Web $sw \in GD_i$

Afin de sélectionner les meilleurs services à intégrer dans un groupe, il s'agit de répondre aux questions suivantes :

1. Est-ce que le service Web a contribué dans le succès d'un groupe critique (*C*), semi critique (*SC*) ou non critique (*NC*) ?
2. Est-ce que le service Web a causé la défaillance d'un groupe *C*, *SC* ou *NC* ?
3. A quel point un service a échoué ou a réussi ses exigences dans le passé ?
4. Existe-t-il un service qui remplace toujours un autre avec succès ?
5. Les services mis à disposition par un même fournisseur travaillent-ils mieux ensemble ?

Il s'agit d'abord de déterminer si un service est bon ou pas, en répondant aux questions précédentes. Ensuite, l'objectif est de déterminer les services Web assurant un meilleur compromis en termes des différents critères. Les n (Algorithme 3) premiers services Web seront sélectionnés pour être impliqués dans le nouveau groupe de diversité.

Le tableau de monitoring illustré en Table 4.6 montre les différentes données à extraire à partir du fichier *Log*. Chaque ligne représente les données collectées pour un service dans le groupe. Chaque service peut être dans un des trois états d'exécutions : {Succès, Valeur, Crash} dans l'intervalle de temps [t_i, t_j]. Les colonnes représentent les données à collecter (e.g., le temps de début et de fin d'un logging pour un service, le degré de confiance des services Web au sein d'un groupe de diversité à un service dans le même groupe et les types d'applications où le service a été exécuté). Chaque case contient le nombre de fois où le service à réussi, échoué ou crashé par exemple.

Formellement le tableau de Monitoring Tab = <x, y, z> sachant que x={ sw_k k=1..nb_{sw} entre [t_i, t_j] / State $\in$ {Succès, Valeur, Crash} }, y= {Data}, z = {z_{ij} | $z_{ij} = x_i.y_j$ }. Une case $z_{ij} = nb^C_{Succes}$ contient le nombre de fois où un service Web sw_i a réussi (state = Succès) dans un groupe critique y_j =*C*. Par exemple, Table 4.6 donne les informations suivantes :

1. Le service sw_1 a été exécuté 30 fois dans l'intervalle de temps [t_0,t_1] (nb_{Exe} = 30).
2. Le nombre d'exécution dans un groupe critique m_c = 11, dont NB_{Succes}=6, NB_{Echec}=2 et NB_{Crash}=3.
3. Le nombre d'exécution dans un groupe semi-critique m_{sc} = 12, dont NB_{Succes}=8, NB_{Echec}=3 et NB_{Crash}=1.
4. Le nombre d'exécution dans un groupe non-critique m_{nc} = 7, dont NB_{Succes}=5, NB_{Echec}=1 et NB_{Crash}=1.
5. Pour un support d'itemset $\mathbb{SIP}_{min}$=2, le sw_3 représente un item fréquent c.-à-d. le service Web sw_3 remplace toujours avec succès le sw_1 dans [t_1, t_2].

À partir de ce tableau, nous pouvons calculer le taux de défaillance d'un service Web sw_i dans [t_j,t_k]. Le taux de défaillance est lié au type d'application, est donné par l'Équation 4.7.

4.6 Principe de sélection de services Web

Après avoir établi le degré de redondance, nous détaillons dans cette section notre méthodologie de sélection de services Web membres ($Select_{SW}$). Cette sélection s'effectue à la base d'un compromis optimal entre les exigences des clients en termes des besoins en sûreté de fonctionnement.

Soit un ensemble de services Web équivalents disponibles qui ont fait partie des groupes de diversité formés précédemment. En tenant compte des exigences actuelles

Table 4.6 – Table de Monitoring

Données / Services	T	$\mathcal{ORD}$	TA / Résultat	C	SC	NC
sw_1	$[t_0, t_1]$	sw_3	Succès	6	8	5
			Valeur	2	3	1
			Crash	3	1	1
sw_2	$[t_0, t_1]$	sw_5	Succès	9	4	2
			Valeur	2	5	1
			Crash	2	3	5
sw_3	$[t_0, t_1]$	None	Succès	6	8	5
			Valeur	2	3	1
			Crash	3	1	1
sw_1	$[t_i, t_j]$	sw_3	Succès	5	3	8
			Valeur	3	3	1
			Crash	3	4	5

des clients, nous cherchons à déterminer l'ensemble des services Web formant le groupe de diversité actuel. Pour cela, nous calculons une liste de valeurs de réussite des services Web disponibles exprimant leur capacité à respecter les critères donnés.

La fonction $Select_{SW}$ est calculée à partir des données collectées (voir Section 4.5). Elle est une agrégation pondérée des paramètres suivants (voir Equation 4.6) :

1. Une valeur de réussite VR. Elle mesure si un service a réussi ou pas les exigences client dans les exécutions précédentes.
2. Un degré de satisfaction DS. Il mesure à quel point le service a réussi ou a échoué dans le passé.
3. Les exigences actuelles du client CR^{act}_{max}={f^{act}_{max}, t^{act}_{max}, r^{act}_{max}}, correspondant respectivement au taux de défaillance maximum toléré, temps de réponse maximum toléré, et le coût de la consommation des ressources acceptable.

$$Select_{SW} = f(VR, DS, CR_j) \tag{4.6}$$

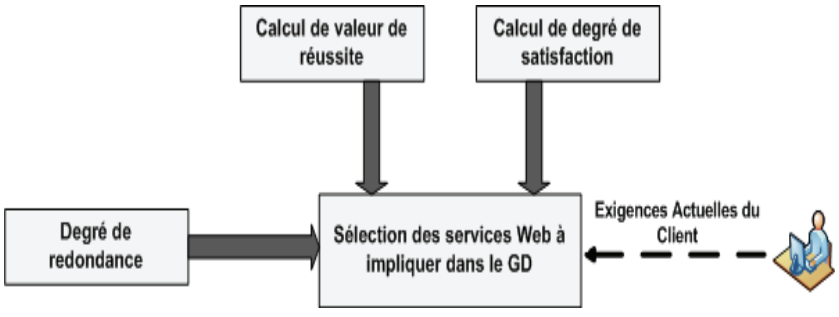


Figure 4.3 – **Processus de sélection des services Web à impliquer dans le GD**

4.6.1 Détermination des valeurs de réussites d'un service Web

A partir des informations collectées présentées dans le tableau de monitoring (Tab 4.6) pour les services Web disponibles, le triplet de valeurs de réussite $VR_i^{CR^{max}_{his}}$ pour quantifier la réussite ou l'échec d'un service sw_i pour les critères CR^{his}_{max}= (f^{his}_{max}, t^{his}_{max}, r^{his}_{max}). Pour chaque critère, nous déterminons la valeur de réussite (i.e., -1, 0.5 ou 1). Un service Web SW_i réussit un critère CR^{his}_{max} si le rapport entre le taux observé pour un critère dans l'historique CR^{his}_{obs} comparé au taux maximum toléré pour un critère dans l'historique CR^{his}_{max} est inférieur ou égal à 1 ($\frac{CR^{his}_{obs}}{CR^{his}_{max}} \leq 1$)

Prenons comme exemple, un service Web qui a été exécuté une fois dans le passé. Le client C de ce service a toléré un taux de défaillance de f^{his}_{max}=0.3. Le rapport entre f_{obs} dans l'exécution précédente et $f^{his}_{max} \in CR^{his}_{max}$ est utilisé pour évaluer la valeur de réussite VR^{f_i}. Le calcul du taux des différents critères observés CR_{obs} dépend du type de groupe de diversité TA sachant que $TA \in \{C, SC, NC\}$ et de la participation d'un service Web au sein d'un groupe $\{C, SC, NC\}$.

Nous spécifions dans la section suivante les cas d'exécutions des services Web dans des GD formés précédemment. Nous prenons comme cas la participation d'un service Web au sein d'un groupe critique C.

4.6.1.1 Cas de participation d'un service dans un groupe

Soit nb_{Exe} le nombre total de participations d'un service Web dans [t_0, t_1], soit m est le nombre de participations de ce service dans un groupe critique. La figure 4.4 montre les différents cas de participation d'un service Web dans un groupe critique. Quatre situations sont possibles (voir Figure 4.4) :

1. Le service Web a été invoqué nb_{Exe} fois mais jamais dans un groupe critique m=0.
2. Le service Web n'a jamais été invoqué dans cette intervalle de temps nb_{Exe}=0.
3. Le service Web a été invoqué que dans des groupes critiques $nb_{Exe} = m$.
4. Le service Web a été invoqué nb_{Exe} fois, dont m fois dans un groupe critique et $nb_{Exe} - m$ fois dans des groupes semi- et non-critiques.

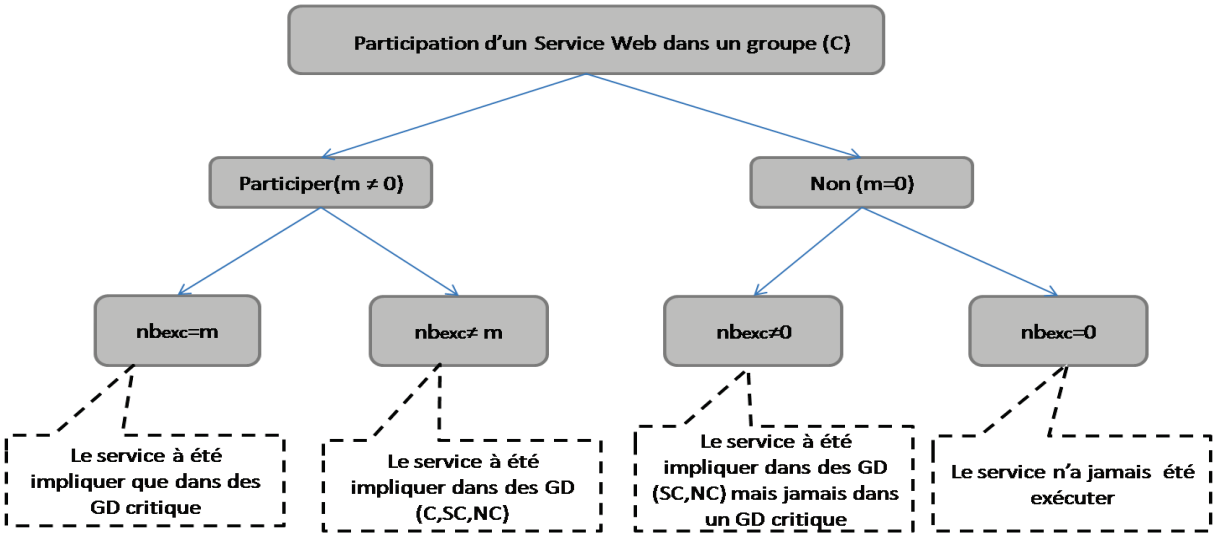


FIGURE 4.4 – **Situation de participation d'un service Web dans un groupe critique**

Le calcul du taux de défaillance d'un service Web dépend de la criticité de la fonctionnalité implémentée par le groupe de diversité dans lequel le service a été impliqué. Ce calcul sera basé sur les constatations suivantes :

- Si la fonctionnalité courante est critique(C), et le service n'a jamais été exécuté dans un groupe critique ($m = 0$), mais exécuté dans un groupe SC ou NC, le taux de défaillance sera calculé en fonction du degré de confiance δ dans ce service donné initialement par le client ($0 < \delta < 1$). Le client peut évaluer ce degré de confiance en se basant sur le comportement du service dans les deux types d'application (SC,NC).
- On doit prendre en considération le nombre de fois où le service n'a pas répondu (NB_{Crash}), à cause d'une indisponibilité, ainsi que le nombre de fois où le service a rendu des résultats inattendus (NB_{Value}).
- Le taux de défaillance ne donne pas une information complète. En effet, lorsque deux services n'ont jamais échoué, il est important d'inclure leur taux de succès.
- Différencier selon les cas et selon le type d'application le taux de défaillance élevé à cause du nombre d'échec ou à cause du nombre de crash. Par exemple, un service lent peut être considéré comme défaillant par crash par rapport à certains utilisateurs exigeants en termes de temps de réponse.

L'équation 4.7 calcule le taux de défaillance d'un service Web qui s'exécute dans un groupe critique $f_{obs}^{his}(sw_C)$ à partir de l'historique.

$$f_{obs}^{his}(sw^C) = \begin{cases} \frac{(NB_{Value} \times w_{Value} + NB_{Crash} \times w_{Crash})^{SC}}{(nb_{Exe_{SC}})} \times \delta, \quad si \quad nb_{Exe} \neq 0 \ \ \& \ \ m = 0 \\ 0, \quad si \quad nb_{Exe} = 0 \\ \frac{(NB_{Value} \times w_{Value} + NB_{Crash})^{C} \times w_{Crash}}{m_C} \times \frac{\sum_{i=0}^{n'} Cf_{sw_{ij}}(sw)}{n'}, \quad si \quad n = m \\ \frac{(NB_{Value} \times w_{Value} + NB_{Crash} \times w_{Crash})^{C}}{m} + \frac{(NB_{Value} \times w_{Value} + NB_{Crash} \times w_{Crash})^{SC} + (NB_{Value} \times w_{Value} + NB_{Crash} \times w_{Crash})^{NC}}{(nb_{Exe} - m)} \\ \times \frac{\sum_{i=0}^{n'} Cf_{sw_{ij}}(sw)}{n'}, \quad si \quad nb_{Exe} \neq m \ \ \& \ \ nb_{Exe} \neq 0 \end{cases} \tag{4.7}$$

δ représente la confiance dans un service qui a réussi dans un groupe non critique et semi critique mais n'a jamais été impliqué dans un groupe critique.

Du même, les taux des différents critères sont calculés selon la participation d'un service Web dans un groupe de type *ta* ou un groupe meilleur. Par exemple le taux de défaillance d'un service Web qui s'exécute dans un groupe *SC* est donné selon sa participation dans les groupes *SC*, mais aussi les groupes *C*.

$$\mathrm{f}_{obs}^{his}(sw^{SC}) = \frac{(NB_{Value}^{C} + NB_{Value_{SC}}) \times w_{Value} + (NB_{Crash}^{C} + NB_{Crash}^{SC}) \times w_{Crash}}{(m + nb_{Exe_{SC}})}$$

Après avoir calculé les différents taux des critères observés CR_{obs}^{his} pour {f_{obs}^{his}, t_{obs}^{his}, r_{obs}^{his}}, la valeur de réussite d'un critère dans le passé exprime à quel point le taux de critère observé est tolérable. Pour quantifier cette valeur, nous calculons le rapport $\frac{CR_{obs}^{his}}{CR_{max}^{his}}$. Un service a satisfait le critère si le rapport est inférieur ou égal à 1. En fonction de ce rapport, nous attribuons à chaque service Web un triplet de valeur de réussite $\mathrm{VR}_i^{CR_{his}^{max}}$ pour les trois critères de sélection (Équation 4.8).

$$VR_i^{CR_{max}^{his}} = \begin{cases} 1 \quad si \ \ sw \ \ reussi \ \ CR \\ -1 \quad si \ \ sw \ \ ne \ \ reussi \ \ pas \ \ CR \\ 0.5 \quad si \ \ nb_{Exe} = 0 \ \ ou \ \ m = 0 \end{cases} \tag{4.8}$$

Notons que *sw* satisfait CR $\in$ {f, t, r} $\Longrightarrow \frac{CR_{obs}^{his}}{CR_{max}^{his}} \leq 1$. Dans ce cas, le service Web reçoit la $VR_i^{CR_{max}^{his}}$=1 pour le critère CR, et $VR_i^{CR_{max}^{his}}$ = -1, sinon.

Si le service n'a jamais été exécuté nb_{Exe}=0, une valeur intuitive est attribuée à $VR_i^{CR_{max}^{his}}$ (i.e., 0.5). Cette valeur donne l'opportunité à ce type de service d'être sélectionné. Cependant, si ce service échoue, il reçoit une pénalité élevée par rapport à un autre type de service où $\mathrm{nb}_{Exe} \neq 0$.

L'algorithme 4 résume les différentes étapes pour calculer la valeur de réussite des services Web prêts à rejoindre le groupe de diversité actuel. Il prend comme entrés, un fichier *Log* contenant les différentes données nécessaires pour effectuer ce calcul *Log*={T, S, R, CR_{max}^{his}, $Cf_{sw_{ij}}$(sw)} et le nombre total d'exécution d'un service dans le passé (nb_{Exe}). Il calcule selon les différents cas spécifiés dans Figure 4.4 la valeur de réussite (VR = $\{VR_i^{CR_{max}^{his}}\}$). La finalité de cette étape est de calculer une liste de valeurs de réussite associe à chaque service, où chaque indice de liste pointe vers un triplet de VR dont les valeurs peuvent prendre (-1,0,5 ou 1).

Algorithme 4: Détermination de $Valeur - Reussite$

Données : Log={T, S, R, $Cf_{sw_{ij}}(sw)$, CR_{max}^{his}}, nb_{Exe}

Résultat : $VR = \{VR_i^{CR_{max}^{his}}\}$

Calculer CR_{obs}^{his} selon Équation 4.7
pour $CR_j \in CR$ **faire**
  si nb_{Exe} =0 **alors**
    $VR_i^{CR_{max}^{his}} = 0.5$ % Le service n'a jamais été exécuté %
  sinon
    suivant $TA \in \{C, SC, NC\}$ **faire**
      cas où C
        si $\frac{CR_{obs}^{his}(C)}{CR_{max}^{his}} \leq 1$ **alors**
          $VR_i^{CR_{max}^{his}} = 1$ % *Le taux* CR_j^p *observé dans un GD critique est acceptable* %
        sinon
          $VR_i^{CR_j^{his}} = -1$
      cas où SC
        si $\frac{CR_{obs}^{his}(C)}{CR_{max}^{his}} \leq 1 \parallel \frac{CR_{obs}^{his}(SC)}{CR_{max}^{his}} \leq 1)$ **alors**
          $VR_i^{CR_{max}^{his}} = 1$ % *Les taux* CR_{obs}^{his} *observés dans un GD critique et semi critique sont acceptables* %
        sinon
          $VR_i^{CR_{max}^{his}} = -1$
      cas où NC
        si $(\frac{CR_{obs}^{his}(C)}{CR_{max}^{his}} \leq 1 \parallel \frac{CR_{obs}^{his}(SC)}{CR_{max}^{his}} \leq 1 \parallel \frac{CR_{obs}^{his}(C)}{CR_{max}^{his}} \leq 1)$ **alors**
          $VR_i^{CR_{max}^{his}} = 1$
        sinon
          $VR_i^{CR_{max}^{his}} = -1$
  Ajouter $VR_i^{CR_{max}^{his}}$ au VR
Retourner VR

4.6.2 Détermination du Degré de satisfaction

Après avoir calculé la valeur de réussite pour chaque service Web disponible, il s'agit dans cette section de déterminer le degré de satisfaction $DS_i^{CR_{max}^{his}}$ des exigences des clients dans des exécutions précédentes. Quand le service SW_i satisfait exactement les exigences des clients, sa valeur de réussite notée $VR_i^{CR_{max}^{his}}$ vaut 1 avec un degré de satisfaction $DSi^{CR_{max}^{his}}$=0.

Ce degré est donné par la formule suivante (voir Equation 4.9) :

$$DS_i^{CR_{max}^{his}} = \begin{cases} 0.5 \quad Si \;\; nb_{Exc} = 0 \\ \frac{CR_{max}^{his} - CR_{obs}^{his}}{CR_{max}^{his}} \quad Sinon \end{cases} \tag{4.9}$$

Sachant que CR_{max}^{his} représente le taux des critères CR maximum toléré extrait à partir du *Log*, $CR_{max}^{his} \in \{f_{max}^{his}, t_{max}^{his}, r_{max}^{his}\}$; $CR_{obs}^{his} \in \{f_{obs}^{his}, t_{obs}^{his}, r_{obs}^{his}\}$ pour un service Web $sw_{ij} \in GD_i$ (j=1..n).

L'algorithme 4.6.2 présente les différentes étapes pour calculer le degré de satisfaction de chaque service Web disponible.

Algorithme 5: Détermination du degré de satisfaction *DS*

Données : nb_{Exe}, CR_{max}^{his}, CR_{obs}^{his}, δ

Résultat : $DS = \{DS_i^{CR_{max}^{his}}\}$: *Liste des degrés de satisfaction des services Web disponibles*

pour $CR_j \in CR$ **faire**
- **si** nb_{Exe} =0 **alors**
  - $DS^{CR_{max}^{his}} = 0.5$
- **sinon**
  - **si** *(m =0)* && *(TA=C)* **alors**
    - $DS^{CR_{max}^{his}} = \frac{CR_{max}^{his} - CR_{obs}^{his}}{CR_{max}^{his}} \times \delta$ % Le service n'a jamais été exécuter dans un GD critique%
  - **sinon**
    - $DS^{CR_{max}^{his}} = \frac{CR_{max}^{his} - CR_{obs}^{his}}{CR_{max}^{his}}$
- *Ajouter* $DS^{CR_{max}^{his}}$ *à DS*

Retourner *DS*

Cet algorithme calcule une liste de degrés de satisfaction attribués à chaque service où chaque indice de liste pointe vers un triplet de *DS* ($DS_i^{f_{max}^{his}}$, $DS_i^{t_{max}^{his}}$, $DS_i^{r_{max}^{his}}$) mesurant le degré de satisfaction des différentes exigences spécifiées selon f, t, r respectivement.

4.7 Fonction de sélection et modèle de pénalité

4.7.1 Fonction de sélection

La fonction de sélection (Equation 4.10) doit assurer un compromis optimal entre les différents critères de sélection qui représentent les exigences des clients. Elle doit prendre aussi en considération les informations extraitres à partir des données collectées. Ces informations sont la valeur de réussite ou pas d'un service Web et le degré de sa réussite ou d'échec de celui ci.

$$Select_{SW} = \left\{ \frac{VR_i^{f^{his}_{max}} \times |DS_i^{f^{his}_{max}}|}{f^{act}_{max}} \times W_f + \frac{VR_i^{t^{his}_{max}} \times |DS_i^{t^{his}_{max}}|}{t^{act}_{max}} \times W_t + \frac{VR_i^{r^{his}_{max}} \times |DS_i^{r^{his}_{max}}|}{r^{act}_{max}} \times W_r \right. \quad (4.10)$$

Sachant que f^{act}_{max}, t^{act}_{max}, r^{act}_{max} représentent les taux des critères actuels largement tolérés.

L'équation suivante 4.11 explique les différents cas où le ratio entre la satisfaction des exigences précédentes et les exigences actuelles est satisfait.

$$\begin{cases} \frac{VR_i^{CR_j} \times |DS_i|}{CR^{act}_{max}} = 1, \ si \ CR_j \ est \ tres \ bien \ satisfait \ par \ sw \\ \frac{VR_i^{CR_j} \times |DS_i|}{CR^{act}_{max}} = 0, \ si \ CR_j \ satisfait \ exactement \ f_{max} \\ 0 < \frac{VR_i^{CR_j} \times |DS_i|}{CR^{act}_{max}} < 1, \ si \ DS_i \ de \ service \ est \ tolerable \\ \frac{VR_i^{CR_j} \times |DS_i|}{CR^{act}_{max}} < 0, \ si \ CR_j \ nst \ pas \ satisfait \ par \ sw \end{cases} \quad (4.11)$$

Les services Web seront classés par ordre descendant de Select_{SW} et seuls les n premiers services avec les plus grandes valeurs seront retenus pour faire partie du groupe. Pour une meilleure sélection les critères doivent être bien pondérés, dans la section suivante, on explique la méthode utilisée pour la pondération basées sur les exigences actuelles des clients.

Algorithme 6: Calcul de l'ensemble LS_{sw}

Données : nb_{Exe}, CR^{act}_{max} : *Les exigences actuelles du client*
Résultat : LS_{sw}=*{(sw, Select$_{sw}$), k=1..n} Liste triée des services Web*
pour *chaque* $CR_j \in CR$ **faire**
  Calcule Poids W_{CR_j}
pour *chaque sw* **faire**
  $VR \longleftarrow Valeur - Reussite$
  $DS \longleftarrow Degre - Satifaction$
pour *j allant de 1 à 3* **faire**
  $\text{Select}_{sw} = \text{Select}_{sw} + \frac{VR_i^{CR^{his}_{max}} \times DS_i^{CR^{his}_{max}}}{CR^{act}_{max}} \times W_{CR_j}$
Ajouter (sw, Select$_{sw}$) à LS_{sw} Trie décroissant de LS_{sw} selon la valeur de Select$_{sw}$
pour *chaque* $sw \in LS_{sw}$ **faire**
  pour *k allant de 1 à n* **faire**
    $GD_i = GD_i + \{sw_k\}$
RetournerLS_{sw}

L'algorithme 6 calcule la valeur de sélection des différents services Web. Il prend comme entrée les exigences actuelles du client CR^{act}_{max} en termes de (f, t, r), les deux listes calculées dans les algorithmes 4.6.2 (page 94)et 4 (page 93) et il retourne une liste ordonnée des services en valeur de sélect.

Le premier service correspond au service avec la plus grande valeur de sélect, le nouveau groupe de diversité implique selon le degré de redondance n calculé dans l'algorithme de calcule du degré de redondance et sa variation (Algorithme 3), les n premiers services Web avec la valeur la plus élevée de sélect.

4.7.2 Modèle de pénalité

Dans cette section nous discutons de l'attribution de pénalité pour les services qui échouent à accomplir leur fonction, et plus particulièrement, les services où on a attribué une valeur de confiance élevée pour le calcule de VR_i et DS_i.

Par exemple, le faite d'attribuer initialement aux services Web qui n'ont jamais étaient exécutés dans le passé, une valeur de 0.5 pour le VR_i et le DS_i, influence considérablement la sélection (comme montre l'expérimentation dans le chapitre 5). Cependant, si ce type de service (nb_{Exe}) échoue à fournir une contribution utile à la procédure de vote dans le cas de protocole parallèle où à satisfaire les spécifications dans le cas du protocole séquentiel, il lui sera associé une dissidence normalisée élevée sous forme de pénalités. Cela permettra d'exclure le service au prochaine tour de sélection.

La valeur $\wp(f_i, t_i, r_i)$ sera calculée en fonction du type de groupe où ce service a été impliqué TA, l'ancien degré de satisfaction DS_i^{t-1} et les valeurs de réussite actuelles VR_i.

Elle est donnée dans l'équation 4.12 suivante :

$$\wp = \prod(DS_i^{t-1}, VR_i, w_{TA}) \tag{4.12}$$

La nouvelle fonction de sélection qui suit le modèle de pénalité est donnée par l'équation 4.13 :

$$Select_{SW_i} = \begin{cases} Select_{SW_i} - \wp & Si \quad \wp > 0 \\ Select_{SW_i} + \wp & Si \quad \wp < 0 \end{cases} \tag{4.13}$$

4.7.3 Attribution des poids de critères

Le calcul de la fonction de sélection de services Web nécessite le calcul du ratio qui évalue à quel point un service a satisfait ces exigences dans le passé, mais aussi le poids de chaque exigence en termes de f_{max}^{act}, r_{max}^{act}, t_{max}^{act}. Par exemple un client définit le taux de défaillance f_{max}^{act} largement toléré, mais la question qui se pose c'est par rapport à quoi ce taux est-il tolérable ? Un client peut tolérer de recevoir une réponse incomplète mais pas une réponse incorrecte. Cela veut dire que chaque critère (exigence) peut être défini selon plusieurs sous critères, par exemple le critère f_{max}^{act} peut être décomposé selon le type de réponse exigé en : réponse correcte, réponse complète, réponse cohérente...etc. La pondération du critère par rapport aux sous critères permet de donner plus de sémantique aux différents critères et au choix du services par la suite.

Thomas Saaty [Saa01] définit une méthode appelée l'analyse hiérarchique multicritères AMH, qui permet la comparaison et le choix entre des options pré-établies. Elle repose sur la comparaison de paires d'options et de critères. Cette méthode repose sur trois concepts :

1. **La structuration hiérarchique** : Il s'agit de déterminer les différents niveaux hiérarchiques et choisir les critères à chaque niveau. Le premier niveau contient les critères principaux, la comparaison s'effectue selon les critères de niveau équivalent. Par exemple à la sélection d'un service, le client peut décomposer son objectif final "sélectionner le meilleur service" en trois éléments comme le bon fonctionnement du service, le temps de réponse du service et le coût du service. De plus ces critères peuvent être décomposés en sous-critères. Ainsi l'utilisateur peut décomposer le critère bon fonctionnement en critères de niveau inférieur comme réponse correcte, réponse complète...etc.

2. **La structuration des priorités** : classement des éléments selon leur importance relative. Il s'agit d'une évaluation de chaque paire de critères. Elle prend la forme de questions successives de type : "Quelle est l'importance du critère A par rapport au critère B ?". Généralement, cinq choix de réponses sont offerts pour évaluer d'une façon numérique l'importance d'un critère par rapport à un autre (importance égale, Modérément plus important, Beaucoup plus important, Considérablement plus important, D'une importance écrasante)

3. **Détermination des poids des critères et sous critères** : Le décideur définit les priorités en comparant de manière binaire les éléments de la hiérarchie. Pour chaque niveau du sommet à la base, il doit comparer les éléments deux à deux par rapport au critère supérieur A. Par exemple, il doit se demander comment l'élément A_1 satisfait-il mieux le critère A que l'élément A_2 (A_1 et A_2 représentent des sous critères de A).

4.8 Conclusion

L'élaboration d'une stratégie de tolérance aux fautes basée sur la diversité des services Web requiert la définition de différents détails concernant le groupe de diversité. Nous avons proposé dans ce chapitre une approche de sélection des services Web les plus fiables à impliquer dans un groupe de diversité. Le processus de sélection est basé sur l'historique d'exécution des services Web. Nous avons distingué les différents cas de participation d'un service Web dans un groupe de diversité pour pouvoir prendre une décision concernant le groupe à former lors d'une invocation.

Dans le cadre de cette approche, nous avons proposé différents algorithmes pour calculer si un service Web a contribué ou pas au succès d'un groupe de diversité, et à quel point il a satisfait les exigences des clients dans le passé. Dans le chapitre suivant nous validons notre travail par quelques expérimentations.

Chapitre 5

Prototype et Expérimentation

Contents

5.1 Introduction

Ce chapitre présente les résultats de faisabilité des différents mécanismes de gestion de fautes supportées par notre architecture de managers, ainsi ceux de l'utilisation de notre approche de sélection dynamique et adaptative de services Web. L'objectif est de montrer à travers des expérimentations le fonctionnement opérationnel de l'architecture et l'efficacité des groupes de diversité obtenus lors de la sélection.

Notre phase d'implémentation est divisée en deux parties. La première concerne le développement des protocoles de tolérances aux fautes incluant les mécanismes d'invocation parallèle et séquentielle, ainsi que la procédure de vote entre des copies équivalentes d'un service Web. La deuxième partie concerne l'implémentation des algorithmes de sélection des services Web à partir d'un fichier Log.

Pour réaliser notre framework de tolérance aux fautes, nous avons développé un ensemble de composants :

- Un service Web manager (WSM) gérant l'exécution de service Web atomique.
- Un groupe manager (MGD) gérant l'exécution de groupe de diversité et les interactions entre les services Web membres de ce groupe.
- Un composite manager (MCS) validant ou annulant l'exécution globale en fonction de la criticité de la fonctionnalité en question.

Ces trois composants ont été développés sous l'environnement de développement *JavaEE* [Micb]. Les services Web sont, quant à eux déployé sous *Glassfish* [Ora]. Nous avons créé nos propres Services Web pour pouvoir injecter des fautes et vérifier l'efficacité des algorithmes proposés.

5.2 Implémentation

Dans cette section, nous présentons les différents outils nécessaires à la mise en œuvre de notre framework ainsi que l'implémentation de ses mécanismes.

5.2.1 Environnement technique

La phase de développement est divisée en deux parties. Dans la première, nous avons utilisé deux catégories de services Web : ceux déployés physiquement sur un serveur d'applications et ceux créés localement. Pour développer une application basée sur les services Web, différents *middlewares* Java existent tels que Aparche Axis [Axi], JBoss [Lab], et Glassfish [Ora] avec des caractéristiques et avantages qui leur sont propres. Nous avons choisi Glassfish pour les raisons suivantes :

- Son environnement de développement ainsi que ses outils sont complètement intégrés dans l'IDE NetBeans.

- Conformité avec les spécifications de services Web et aux normes d'interopérabilité de services Web.
- Projet open-source avec un fort soutien industriel à la fois par *SunMicrosystems* et *Microsoft*.

Par la suite, nous définissons l'environnement de développement et les bibliothèques nécessaires pour mettre en œuvre notre framework :

- Éditeur IDE : Netbeans v 7.3 [Net]
- Java SDK : J2EE v1.6
- Processes Intel(R)core(TM)i5 - 4GB RAM
- Plateforme des services Web : Glassfish 3.1.2
- API Java des services Web : JAX-WS [Com], JAX-RPC [Mica] et SAAJ [Topo3].

5.2.2 Mécanismes de gestion de fautes

Rappelons que les modèles d'exécution représentent la séquence d'actions déclenchées par les managers afin de répondre à un évènement particulier. Selon le mode d'activation choisi (Parallèle ou Séquentiel), différentes fonctions sont implémentées pour chaque niveau d'abstraction (service Web, groupe, et composition). A chaque niveau est rattaché un composant logiciel dans la composition de services pour assurer son fonctionnement.

Le *MSW* (Web Service Manager) est associé au premier niveau de service concrets. Ce dernier fait référence à tous les services pouvant être mis à disposition des utilisateurs (Service Web disponibles). Cet ensemble de services peut évoluer dans le temps. Le niveau groupe de diversité est rattaché à un composant appelé groupe manager (*MGD*). Chaque groupe est défini par un service abstrait et un ensemble des services concrets similaires. Le nombre des services concrets dans un groupe de diversité peut lui aussi varier de manière dynamique. Le dernier niveau rattaché au composant logiciel *MCS* est constitué d'un nombre quelconque de groupes de diversité.

Quand un client émet une requête, un évènement **Appeler()** se produit entre le client et le manager du composite *MCS*. Cet évènement déclenche l'action invoquée en exécutant le premier groupe de diversité. Lors de l'activation du groupe de diversité par le *MGD*, deux fonctions peuvent être appelées, soit **ActiveP()** pour le parallèle, et **ActiveS()** pour le séquentiel.

Dans le cas du parallèle, le *MGD* intercepte les réponses des services Web jusqu'à la réception d'une majorité. Il déclenche alors la procédure de vote, et détecte les éventuelles fautes. Dans le cas séquentiel, il vérifie le résultat renvoyé par le service à l'aide des spécifications données par l'utilisateur. Le *MSW* se charge d'observer l'état du service Web atomique et d'envoyer l'état de succès ou d'échec au *MGD*.

Nous détaillons les principales fonctions mises en œuvre dans notre implémentation afin d'assurer la continuité de l'exécution des services Web.

5.2.2.1 La fonction : ActiveteP()

Ce type de recouvrement est très utile dans le cas où on a une contrainte forte de temps. Il peut traiter les défaillances Byzantines s'il est doté par un mécanisme de vote ou de comparaison avec des spécifications données par le client sur le résultat attendu. Pour cela, différentes répliques équivalentes peuvent être utilisées et activées en parallèle. D'un point de vue *implémentation*, cela se traduit par la création d'un thread pour chaque point d'accès. Chaque thread se charge de traduire la requête pour la faire correspondre au point d'accès de la réplique.

Lors de la réception de réponses, le thread insère le message correspondant dans un fichier dédié à la collection des différentes réponses (ou résultats). Selon le mode de sélection du résultat et les modèles définis dans le chapitre 3, les réponses de ces services seront analysées par le manager correspondant et renvoyées au client dans un message `SOAP`. Le message `SOAP` sera alors stocké sous format d'un fichier `XML`, pour qu'il puisse être parsé afin de récupérer le résultat correspondant.

5.2.2.2 La fonction : ActivateS()

Ce type de recouvrement est utile dans le cas où il n'existe pas de contraintes sur le temps de réponse, et le coût ainsi que la consommation de ressources doivent être minimisés. La fonction ActivateS() consiste à sélectionner le service Web primaire parmi les services Web disponibles dans le groupe de diversité. La détection d'erreurs est réalisée au moyen de chien de garde (*Watchdog*) ou de méthodes de comparaison avec une spécification donnée. Dans ce modèle d'exécution (Figure 3.7), une liste de répliques est créée. Cette liste est passée en revue en envoyant la requête successivement à chacun jusqu'à l'obtention d'une réponse correcte ou l'épuisement de la liste des services Web disponibles.

La figure 5.1 montre une partie du code Java pour la construction d'un message `SOAP` contenant l'opération à invoquer et les paramètres associés. Il indique aussi la façon d'établir une connexion avec le service Web primaire du *GD*, et d'envoyer le message `SOAP` à ce service.

5.2.2.3 La fonction : Vote()

Dans le cas parallèle, le modèle d'exécution avec la fonction "Vote()" concerne essentiellement des services fournissant des résultats critiques, sur lesquels nous souhaitons tolérer des fautes de type Byzantine.

La requête est envoyée à toutes les répliques disponibles pour être traitée. Ensuite, il s'agit de collecter une majorité pour pouvoir effectuer le vote. Le *MGD* aura besoin de $2*f+1$ réponses pour tolérer f fautes de type Byzantine. Quand un vote majoritaire n'est

```
import javax.xml.soap.*;
...
public void activateS (String urlLeader, String Op, String Param)
{
MessageFactory factory = MessageFactory.newInstance();
SOAPMessage msg = factory.createMessage();
SOAPPart soapPart = msg.getSOAPPart();
SOAPBody body = msg.getSOAPBody();
QName bodyName = new QName(urlLeader,Op,Param);
SOAPBodyElement bodyElt =body.addBodyElement(bodyName);
soapConnectionFactory =SOAPConnectionFactory.newInstance();
SOAPConnection connection =
soapConnectionFactory.createConnection();
java.net.URL endpoint = new URL(urlLeader+"/quotes");
SOAPMessage resp =connection.call(msg, endpoint);
connection.close();
}
```

FIGURE 5.1 – **Invocation dans le groupe manager** *MGD*

pas possible, le *MGD* envoie au manager du composite *MCS* un message avec un état **Échec** du groupe.

5.2.2.4 La fonction : SelectWS()

Cette fonction nous permet de choisir les services Web à impliquer dans le groupe de diversité actuel. Nous avons implémenté deux méthodes de sélection des services Web. La première est basée sur le choix de critères de *QoS* (coût d'invocation de service, réputation de service, rapidité de service) ou même des qualités sur le résultat rendu ; par exemple un client invocant un service *ReservationVol* choisit le service avec le prix le moins cher. La deuxième méthode est basée sur la sélection des services Web à partir d'un log contenant une trace d'exécution de ces services dans le passé.

Dans la première méthode de sélection, le client choisit d'abord un critère de *QoS*. Le *MGD* recherche parmi les services Web disponibles, ceux répondant au critère demandé. Puis il invoque ces derniers comme suit :

– **Dans le cas du protocole Séquentiel :** Le *MGD* sélectionne d'abord le service Web devant jouer le rôle du leader. Si ce service ne réussit pas c.-à-d. ne répond pas (*défaillance franche*) ou rend des résultats incorrects (*défaillance par valeur*), un nouveau leader est sélectionné parmi ceux satisfaisant le critère. La requête sera alors redirigée vers ce nouveau leader pour traitement. La procédure de sélection continue tant que la requête n'est pas satisfaite et il existe encore des services Web

équivalents disponibles.

– **Dans le cas de protocole Parallèle :** Le *MGD* sélectionne les n premiers services selon le critère de sélection. Si le groupe n'arrive pas à décider d'un résultat commun, un autre groupe sera sélectionné. Les algorithmes présentés dans le chapitre 4 ont été implémentés pour réaliser cette sélection. À partir d'un fichier Log où des informations concernant les services sont collectées, une valeur de réussite VR est calculée pour chaque service Web en fonction des exigences clients données dans le passé. Un degré de satisfaction DS est aussi calculé. Les services sont alors triés par ordre décroissant les valeurs de sélection calculées à partir des VR et DS.

5.3 Scénario d'exécution des services Web à l'aide de notre plateforme

La figure 5.2 montre le diagramme de classes associé à la mise en œuvre de notre framework. Ce diagramme résume les principales classes de notre implémentation ainsi que les méthodes les plus importantes.

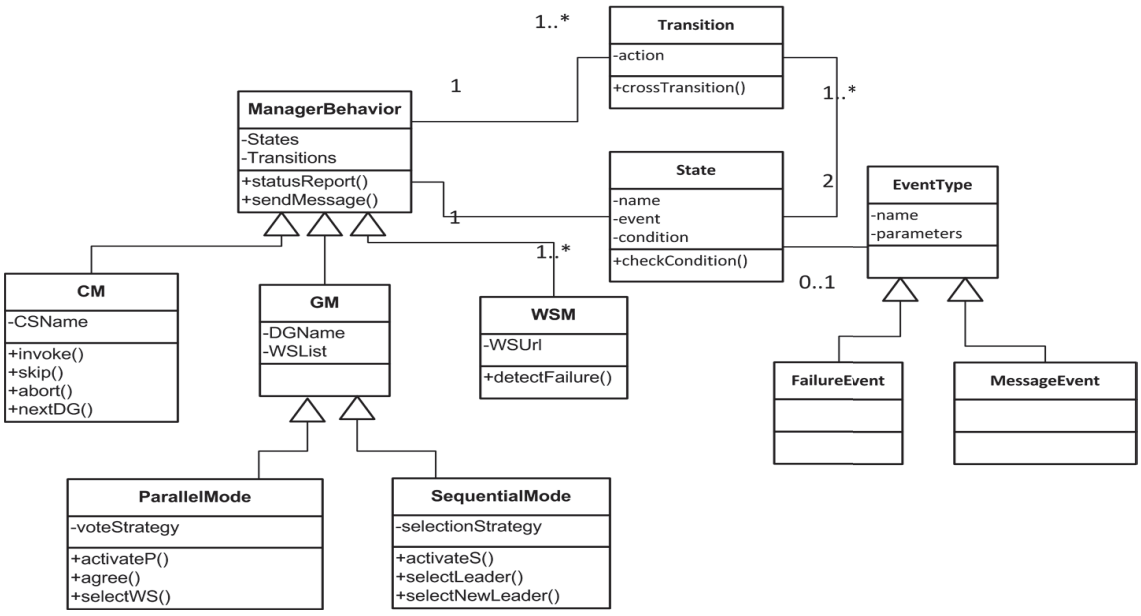


FIGURE 5.2 – **Diagramme de classe de la solution**

Les classes pertinentes correspondent aux *CM*, *GM*, *WSM* implémentant le fonctionnement des trois managers *MCS*, *MGD* et *MSW* respectivement. Chaque classe simule le comportement d'un ou plusieurs automate(s) à états finis caractérisée par des états et des transitions entre ces états. Toutes ces classes héritent de la classe *ManagerBehavior*. Cette dernière décrit les méthodes nécessaires pour le transfert d'états entre les managers.

Il existe deux types d'évènements : *EvenementdEchec* concerne les évènements pouvant survenir au moment de l'activation ou de l'exécution du service (e.g., *CrashServeur*()), et les évènements *EvenementsMessages* exprimant l'exécution d'actions comme *Skip*(). La classe *ManagerBehavior* permet de reporter l'état d'un manager vers un autre ainsi que le résultat obtenu :

Le manager du composite *MCS* implémente les méthodes suivantes : i) *Invoque*() pour déclencher l'exécution d'un groupe de diversité ; elle sélectionne le groupe passé en paramètre et envoie la requête au groupe ; ii) *Valid*() pour valider une réponse correcte ; *Next*() pour sélectionner un nouveau groupe ; iii) *Abort*() pour annuler le groupe selon la criticité de la fonctionnalité ; iv) *Skip*() pour ignorer le groupe en cas d'échec ; et v) *Echec*() pour envoyer l'état échec par le *MGD* lorsqu'il ne reste aucun groupe à invoquer.

Le manager de groupe de diversité *MGD* est le cœur de notre implémentation car il réalise les méthodes permettant d'assurer la continuité d'exécution des services en dépit de fautes.

1. La classe *ParallelMode* implémente i) la méthode *ActivateP*() pour invoquer un nombre défini des services Web en parallèle ; ii) la méthode *Agree*() réalisant le vote des résultats provenant des services Web équivalents ; et iii) la méthode *SelectWS*() pour choisir les services Web à impliquer dans le groupe en diversité courant.
2. La classe *SequentielMode* implémente principalement la méthode *ActivateS*() pour activer le premier service dans le groupe respectant le critère de sélection selon les deux méthodes de sélection mentionnées dans la section précédente. Si le premier service échoue, un autre service sera activé d'une manière automatique jusqu'au l'obtention d'une réponse ou l'épuisement des services Web disponibles.

Le manager de service Web atomique *MSW* implémente la méthode d'invocation d'un service Web atomique (*Activate*()) et la méthode de détection des fautes (*Detect*()). Il envoie la requête au service Web sélectionné, attend la réponse et effectue la détection locale si la requête possède une spécification, sinon il renvoie la réponse au manager du groupe pour effectuer un vote.

5.4 Expérimentation

Le but de cette section est de montrer que la stratégie adaptative de sélection des services Web améliore l'efficacité du groupe comparée à des stratégies statiques. Pour valider notre approche, nous avons effectué plusieurs expérimentations afin d'évaluer

les différents paramètres tels que l'impact de valeur de confiance sur la sélection des services Web, ainsi que l'impact d'attribution de pénalité sur les services Web.

Tout d'abord, il s'agit de vérifier la validité de la sélection des membres de services à impliquer dans le groupe de diversité conçu initialement. Ensuite il s'agit de mesurer l'impact de certains paramètres sur la sélection des services Web et de montrer à travers l'expérimentation l'efficacité de la sélection adaptée basée sur le *Log* par rapport à la sélection basée sur les critères de *QoS*.

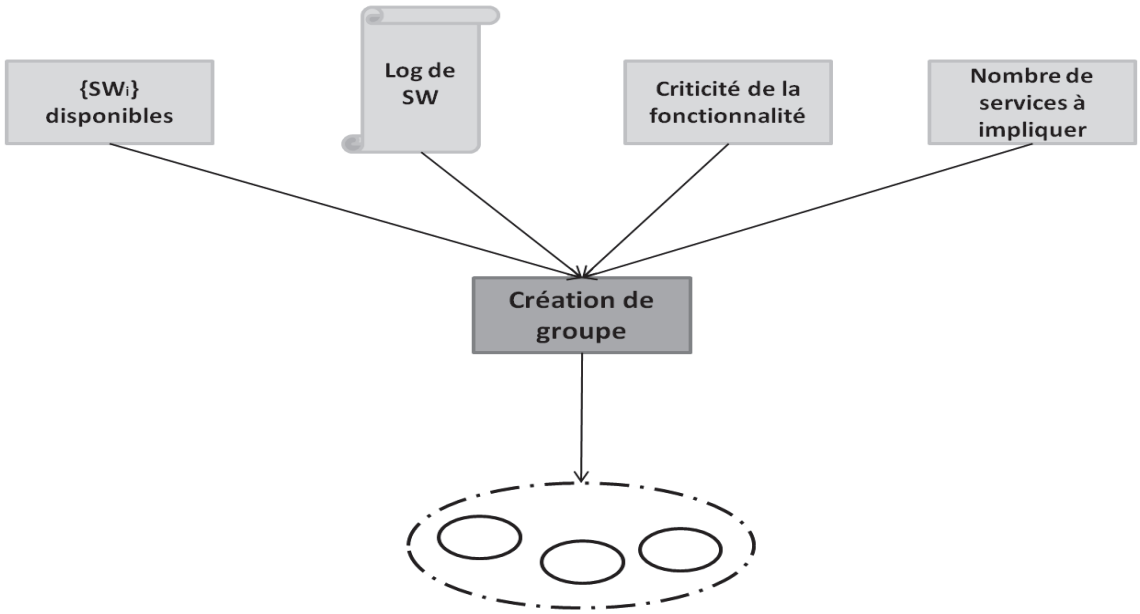


FIGURE 5.3 – **Étape d'implémentation**

5.4.1 Création d'un groupe

Validité du groupe initial : Le but est de montrer que la variation du nombre de services à impliquer dans le groupe n'influence pas les services Web à impliquer, et que les n meilleurs services avec les meilleures valeurs de sélection seront toujours les services à intégrer dans le *GD* initial à un instant donné. Le *MGD* possède les URI de tous les services à invoquer. Lors de l'invocation de groupe de diversité, le *MGD* lance l'exécution des n premiers services sélectionnés suivant la stratégie de recouvrement choisie. Les résultats de cette expérience sont donnés dans la figure 5.4 et concernent la création du groupe de diversité initial. Pour effectuer les tests, nous avons comparé les services disponibles en considérant les conditions suivantes :

- le taux des critères largement tolérés par les clients à f^{act}_{max} (taux de défaillance largement toléré), t^{act}_{max} (le délai de temps largement accepté), r^{act}_{max}(La consommation de ressource largement toléré)}sont fixés à {0.2, 0.3, 0.3} respectivement.
- la valeur de confiance placée à un service ayant participé au succès d'un groupe *SC* et *NC* mais n'a jamais été évaluer dans un groupe *C* est fixée à δ= 0.1.

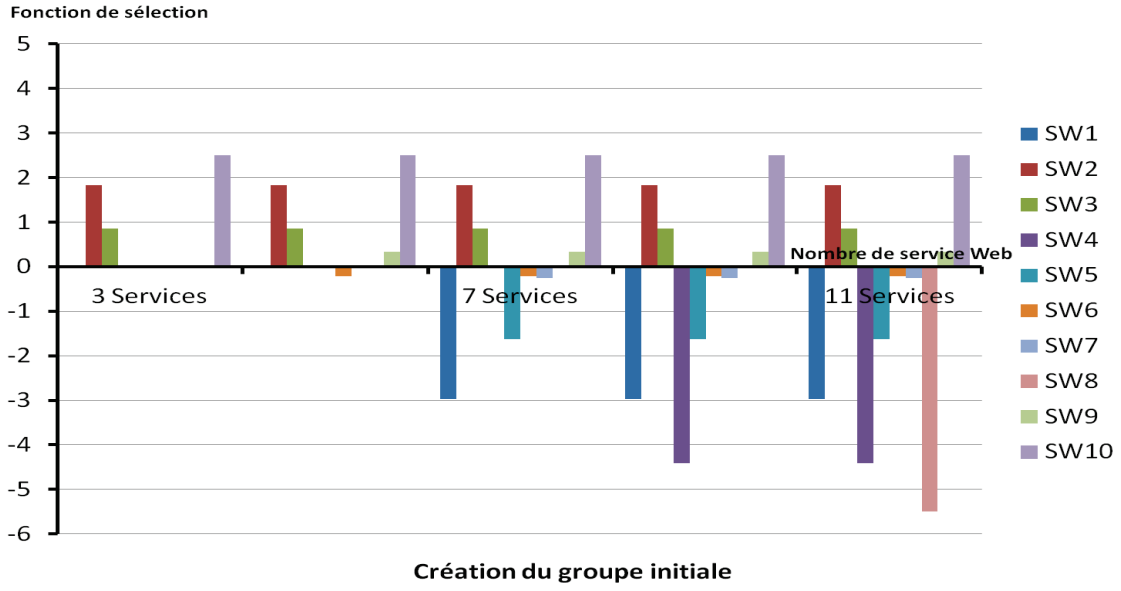
– le nombre des services à impliquer dans le groupe de diversité est variable.

Le tableau suivant 5.1 résume les différentes fonctions de sélection calculées à partir des paramètres mentionnés auparavant. Nous lançons la procédure de calcul plusieurs fois sur l'ensemble des services Web équivalents disponibles (initialement 10 services). Comme le montre le tableau 5.1 le service Web SW_{10} possède la valeur de sélect la plus élevée. Il a satisfait les différentes exigences dans le passé ainsi que les exigences actuelles.

TABLE 5.1 – Résultats de monitoring des services Web

Services Web	VR_i	DS_i	Select
SW_1	(-1.0 , 1.0, -1.0)	(-0.93, 0.0, -2)	-2.900
SW_2	(1.0 , 1.0, -1.0)	(1.0, 0.0, -2)	1.833
SW_3	(0.5 , 0.5, 0.5)	(0.5, 0.5, 0.5)	0.858
SW_4	(-1.0 , -1.0, 1.0)	(-1.699, -0.5, 0.4)	-4.412
SW_5	(-1.0 , 1.0, 1.0)	(-0.78, 0.0, 1.0)	-1.631
SW_6	(-1.0 , 1.0, 1.0)	(-0.012, 0.025, 0.075)	-0.210
SW_7	(-1.0 , -1.0, 1.0)	(0.0, -0.025, 0.04)	-0.251
SW_8	(-1.0 , -1.0, 1.0)	(-1.87, -1.5, 0.25)	-5.492
SW_9	(1.0 , 1.0, 1.0)	(0.1006, 0.0, 0.25)	0.334
SW_{10}	(1.0 , 1.0, 1.0)	(1.0, 0.0, 0.0)	2,5

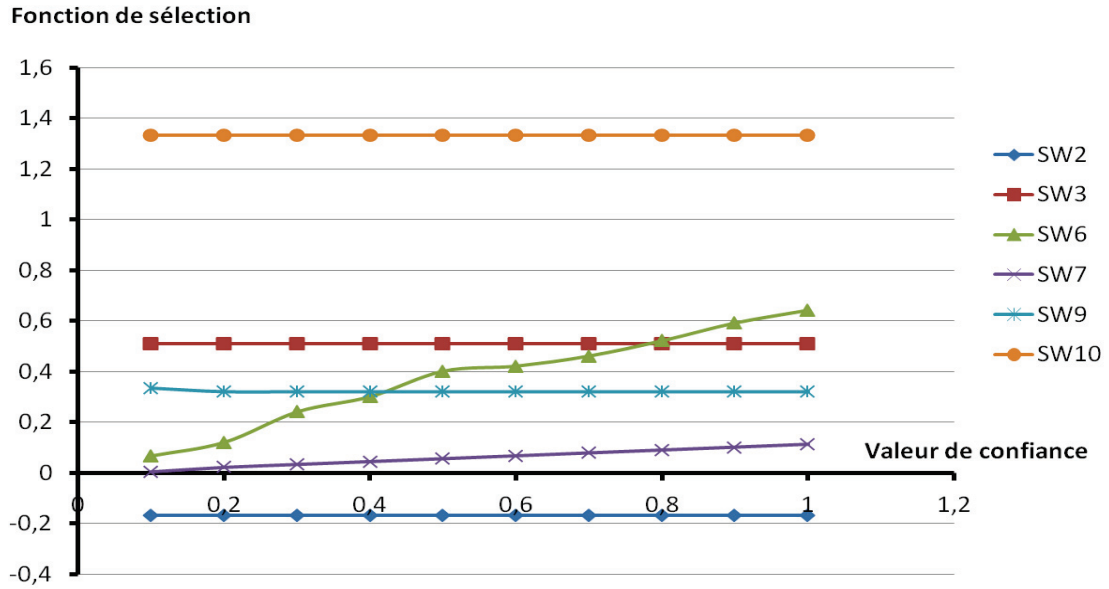
FIGURE 5.4 – **Création de groupe initial**



Impact de la valeur de confiance sur la sélection des services Web : Dans cette expérimentation (Figure 5.5), nous mesurons l'impact de la valeur de confiance accordée par les clients à des services Web, sur la fonction de sélection. Cette valeur affecte la sélection des services Web, dans le sens où un service n'ayant jamais été exécuté

dans un groupe critique, lui est associé un degré de sélection calculé en fonction des taux d'échec et réussite dans un groupe semi critique. Comme cette valeur est donnée par les clients, il est important d'étudier son impact sur la sélection. Nous pouvons observer que la valeur de sélection des services SW_6 et SW_7 où $m = 0$ augmente proportionnellement avec l'augmentation de la valeur de confiance placée dans un service exécuté dans un groupe semi critique.

FIGURE 5.5 – **Impact de valeur de confiance**

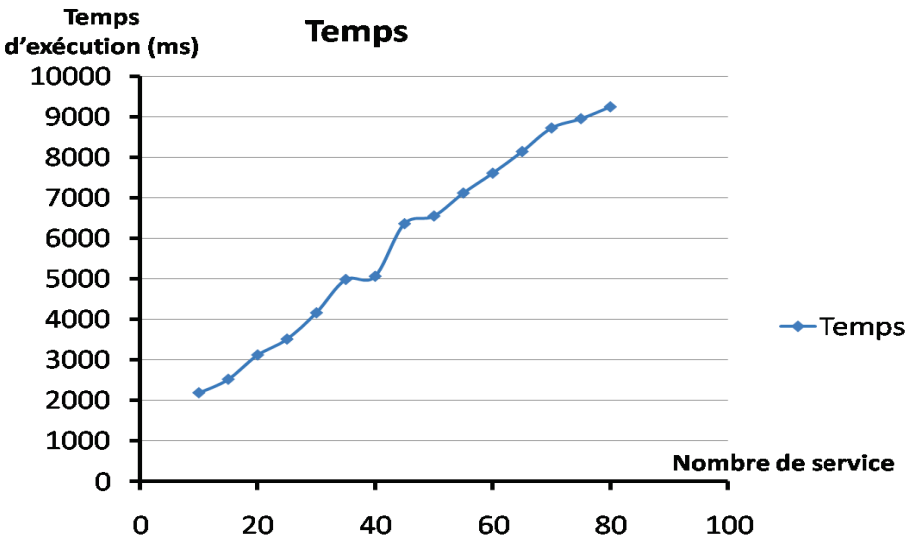


5.4.2 Exécution d'un groupe

Performance. Pour dérouler les expériences suivantes, nous avons utilisés des Services Web crées localement. Différents tests ont été réalisés pour mesurer les performances en termes de temps d'exécution et de sélection des services Web. La Figure 5.6 montre les résultats obtenus en variant le nombre des Services Web utilisés renvoyant différent résultats. Nous mesurons le temps d'exécution et de sélection des différents services Web. La figure montre que le temps de calcul et d'exécution des services Web varie de manière linéaire. Ceci permet de constater un passage à l'échelle de notre approche de sélection.

Efficacité du modèle de pénalité. Il s'agit de montrer l'efficacité de notre modèle de pénalité associé aux services Web défaillants. Nous sélectionnons cinq services (n=5) possédant les meilleures fonctions de sélection, les groupes conçus peuvent être exécutés dans différents type d'application (C, SC, NC). Pour cela des fautes sont injectées dans des services bien ciblés pour montrer dans quel cas de service

FIGURE 5.6 – **Temps d'exécution et de sélection des services Web**



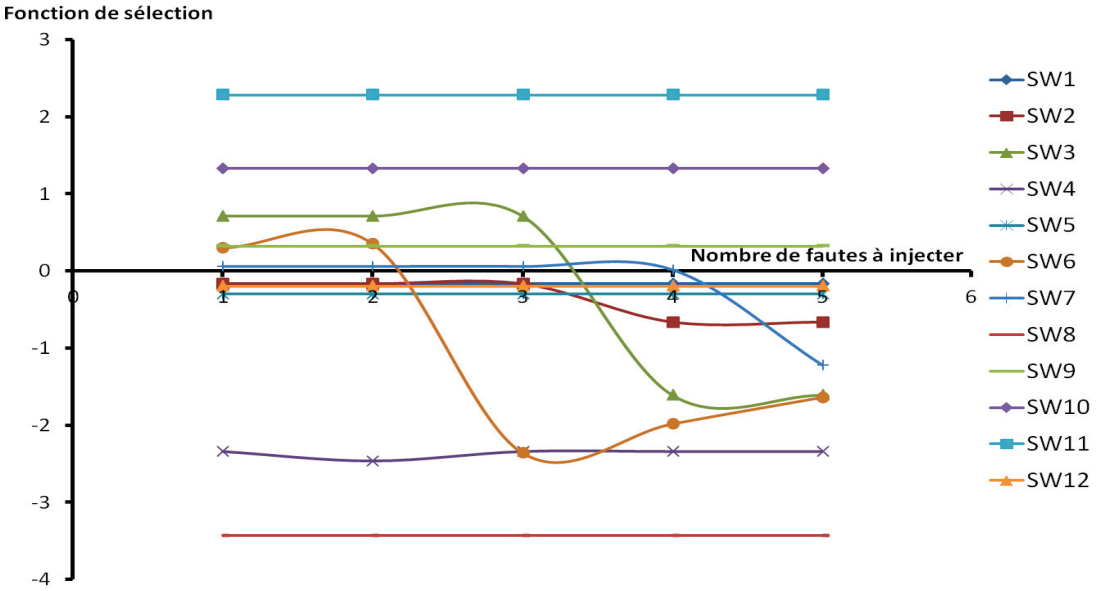
la pénalité influence la sélection. Ces cas de service sont spécifiés dans le chapitre 4 (Figure4.4). Les différents cas testés sont définis comme suit :

1. Aucune faute n'est injectée dans les services (cas initial).
2. Une faute par valeur est injectée dans le SW_6 ; ce service n'a jamais été exécuté dans un groupe critique mais il a réussi toutes les exécutions dans un groupe semi critique. La faute est injectée lors de participation du service dans un groupe semi-critique (*SC*).
3. Une faute par valeur est injectée dans le SW_6 participant dans un groupe critique.
4. Une faute par valeur est injectée dans le SW_3 ; ce service n'a jamais été exécuté (nb_{Exe}=0). La faute est injectée lors de participation du service dans un groupe critique.
5. Une faute par valeur est injectée dans le SW_7 ; ce service n'a jamais été exécuté dans un groupe critique, mais il a réussi toutes les exécutions dans un groupe non critique. La sélection de ce service dans les exécutions précédentes est due au fait de placer un degré de confiance assez grand dans ce type de services. La faute est injectée lors de participation de ce service dans un groupe critique.

La figure 5.7 montre les résultats obtenus pour le modèle de pénalité $\wp$. Nous remarquons que les valeurs de fonction de sélection pour les services non injectés par des fautes sont constantes. La première exécution sélectionne le groupe : { SW_{11}, SW_{10}, SW_3, SW_9, SW_6} comme aucune faute n'est injectée. Nous remarquons

que le SW_3 n'ayant jamais été exécuté nb_{Exc}=0, est sélectionné en troisième position après les services SW_{11} et SW_{10} qui satisfont tous les critères. Après l'injection de fautes par valeur dans le service SW_6 exécuté dans un groupe critique et dans un groupe semi critique, la valeur de sélection se dégrade car il reçoit une pénalité de $\wp = -0.027$ à cause de la faute injectée dans le groupe SC. Lors d'injection de fautes par valeur dans le service Web SW_6 exécuté dans un groupe critique, ce dernier reçoit une pénalité de $\wp = 1.166$. Par conséquent sa valeur de sélection se dégrade considérablement. Dans la prochaine sélection, le service sera écarté malgré la valeur de confiance assez élevée donnée par le client à ce type de service. Dans la quatrième exécution nous injectons une faute par valeur dans le SW_3. Cette faute affecte considérablement la sélection de SW_3. Ceci s'explique par le fait que le modèle utilisé pénalise plus les services dans lesquels une grande confiance est placée (cas de service n'ayant jamais été exécuté nb_{Exc}=0).

FIGURE 5.7 – **Sélection des services après attribution de pénalité sur les défaillances**



Comparaison entre sélection à base du *Log* **et** *QoS*. Dans cette expérimentation, nous comparons notre approche de sélection des services Web divers à impliquer dans un groupe de diversité basée sur le log, avec une approche de sélection des services Web basée sur les paramètres de *QoS* et plus particulièrement la réputation des services au sein d'une communauté de service Web [WZV10].

Nous prenons un même ensemble des services Web disponibles implémentés localement. Cet ensemble contient des services n'ayant jamais été exécutés, ceux n'ayant jamais été exécutés dans un groupe critique et ceux ayant participé dans différents exécution avec différents types de groupe. Nous avons implémenté un

simulateur de pannes, qui choisit un service de façon aléatoire pour injecter une faute par valeur dans le service.

Dans le chapitre 4, nous avons montré comment calculer la fonction de sélection des services Web disponibles à partir de la valeur de réussite VR et le degré de satisfaction DS d'un service. Ce calcul est effectué tenant compte les exigences passées et exigences actuelles d'un client donnée. Une mesure très importante pour évaluer la fiabilité des groupes construits par notre approche est la qualité de groupe conçu QoG. Pour pouvoir comparer les deux approches de sélection (QoS, Log), nous calculons d'une part la qualité de groupe définie en fonction des mesures de fiabilité, disponibilité et pertinence des services Web dans un groupe (approche Log). D'autre part, nous calculons la QoG définie en fonction de la réputation totale calculée après la réception des valeurs de réputation d'un service Web à partir de tous les services Web fiables. Nous utilisons les deux approches de sélection pour former les différents groupes, selon le mode d'activation parallèle. Nous présentons dans le tableau 5.2 les résultats relatifs aux services Web sélectionnés et à l'efficacité des groupes formés :

- La première colonne contient le nombre de services Web à sélectionner pour traiter une requête.
- Dans le cas de sélection à base de QoS (réputation), les colonnes 2 et 3 récapitulent les membres de services Web sélectionnés au sein d'un groupe de diversité.
- Dans le cas de sélection à base du Log, les colonnes 4 et 5 récapitulent les membres de services Web sélectionnés au sein d'un groupe de diversité.

TABLE 5.2 – **Groupes formés avec leurs valeurs d'efficacité**

Nombre du services	**Sélection à base du QoS (Réputation)**		**Sélection à base du Log (*RWSSA*)**	
	GD formé	QoG	*GD* formé	QoG
3	(S_5,S_3, S_9)	0.60	(S_{11},S_{10}, S_3)	0.78
5	(S_5,S_3, S_2, S_4, S_6)	0.53	(S_{11},S_{10}, S_3,S_9,S_6)	0.68
7	(S_5,S_3, S_2, S_4, S_6,S_1,S_{11})	0.32	(S_{11},S_{10}, S_3,S_9,S_6,S_7,S_5)	0.45

La figure Fig.5.8 montre une comparaison entre les deux approches de sélections des services Web pour mesurer l'efficacité de notre approche comparée à l'approche de sélection à base de paramètres de QoS tels que la réputation. Les groupes conçus avec notre approche présentent une meilleure qualité de groupe car la sélection est basée sur les informations collectées concernant les services Web.

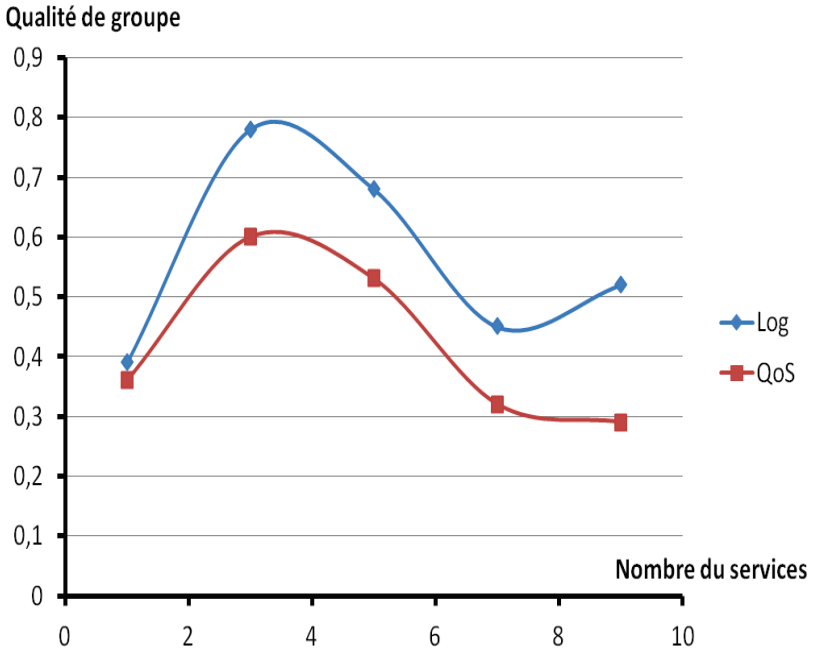


FIGURE 5.8 – **Comparaison entre les approches de sélection à base du Log et de QoS**

5.5 Conclusion

Dans ce chapitre, nous avons présenté les divers composants de notre implémentation. Nous nous sommes essentiellement concentrés sur les défaillances par valeur et Byzantines afin de pouvoir vérifier l'efficacité de notre approche de sélection.

Nous avons fourni, via notre framework développé, des protocoles spécifiques de tolérance aux fautes permettent d'exécuter et de coordonner les services Web au sein d'un groupe, pour répondre aux besoins de la fonctionnalité implémentée par ce groupe.

Enfin, nous avons examiné à travers des expérimentations l'efficacité des groupes créés par la stratégie adaptative comparée aux stratégies statiques. Les résultats obtenus lors des différentes expériences effectuées ont permis de justifier l'intérêt de sélectionner des services à base du Log.

Chapitre 6

Conclusion et Perspectives

Contents

6.1 Conclusion

Les Services Web constituent la technologie de base pour le développement d'Architectures Orientées Services. Ces architectures, de plus en plus répandues, permettent de mettre en place des applications critiques telles que celles relatives au contrôle aérien et au domaine militaire. Elles se basent sur la notion de relation de "service" formalisée par un contrat qui unit le client et le fournisseur de services. Ce type d'application nécessite une haute sûreté de fonctionnement. Celle-ci est définie comme la caractéristique qui vise à placer une confiance justifiée dans les services offerts par une application.

Une architecture multi niveaux pour la tolérance aux fautes des services Web. Dans cette thèse nous avons défini une architecture multi niveaux pour la tolérance aux fautes de services Web. Nous avons introduit la notion des managers spécifiques à chaque niveau d'abstraction des services Web. Ces managers ont le rôle de surveiller chaque niveau : composant, composite et groupe de diversité. ils implémentent différentes techniques de détection et de recouvrement d'erreurs qui sont déclenchées quand les services Web ne satisfont plus les caractéristiques de sûreté demandées. Le comportement des managers dédiés aux différents niveaux de l'architecture proposée est modelisé par des automates à états finis. La particularité de notre approche réside dans l'utilisation de la notion de diversité de services Web, une diversité qui désigne des services Web

possédant des descriptions *WSDL* différentes mais capables d'offrir la même fonctionnalité. Ces services sont regroupés au sein d'un même espace virtuel nommé groupe de diversité. Ce groupe est doté de protocoles de tolérance aux fautes qui représentent les stratégies de recouvrement dans le cas d'activation des services en mode séquentiel ou en mode parallèle.

Adaptation des protocoles de recouvrement. Les deux protocoles séquentiel et parallèle ont été adaptés au contexte de groupes de diversité de services. Le premier protocole (Séquentiel) permet l'invocation d'un seul service à la fois, celui-ci est sélectionné sur des critères fonctionnels (liés aux fautes) ou non fonctionnels (QoS). Si le service invoqué ne retourne aucun résultat (défaillance de type crash) ou retourne des résultats qui ne correspondent pas aux spécifications de l'application (défaillance de type Byzantine), un autre service est sélectionné dans le même groupe de diversité pour être invoqué. Ce processus continue jusqu'à l'obtention d'un résultat sans fautes ou épuisement du groupe de diversité.

Le deuxième protocole (Parallèle) permet l'exécution concurrente de plusieurs services d'un même groupe. Ces derniers sont aussi sélectionnés sur des critères fonctionnels ou non fonctionnels. Un seul résultat sera retourné au client ayant invoqué la fonctionnalité. Le choix du résultat est associé à un algorithme de vote qui se base sur la réception d'une majorité de réponses pour effectuer un vote. Nous avons proposé un algorithme de vote sur le résultat à retourner qui prend en compte la déviation fonctionnelle acceptée entre les services Web équivalents.

Configuration des groupes de diversité. L'élaboration d'une stratégie de diversité requiert la définition de différents détails comme le nombre nécessaire de services Web divers, l'emplacement de ces services Web et les services Web permettant d'assurer une meilleure qualité du groupe de diversité. Dans ce but, nous avons calculé le degré de redondance nécessaire pour chaque protocole (Parallèle et Séquentiel). Le redimensionnement d'un groupe de diversité permettra d'ajuster automatiquement le degré de redondance noté n à utiliser pour s'accorder à l'évolution des fautes observées (Nombre de fautes et type de fautes).

Sélection des meilleurs services dans un groupe de diversité. Nous avons défini une approche de sélection des services Web équivalents à impliquer dans un groupe de diversité. Cette sélection est basée sur l'historique de services Web et les exigences actuelles d'un client donné. Dans le protocole parallèle, la sélection des n meilleurs services Web est faite en fonction du degré de satisfaction des exigences clients vis-à-vis des

services Web. Dans le protocole séquentiel, une réservation des n meilleurs services est nécessaires dans des cas critiques, néanmoins, une détermination de l'ordonnancement de ces services Web est indispensable pour éviter des exécutions inutiles. Pour cela, nous avons utilisé la notion d'itemset fréquent dans un fichier Log où une séquence d'exécution des services Web est considérée comme étant un ensemble d'items. Si la fréquence d'apparence d'une exécution est supérieure à un certain seuil, l'exécution est considérée comme fréquente.

Prototypage et Expérimentation. L'approche proposée a été implémentée sous forme d'un prototype de tolérance aux fautes. Dans un premier temps nous avons mis en place les protocoles de tolérances aux fautes (l'invocation parallèle et l'invocation séquentielle ainsi que la procédure de vote entre des copies équivalentes de services). La deuxième partie concerne l'implémentation des algorithmes de sélection des services Web à partir d'un fichier Log. Nous avons examiné à travers des expérimentations l'efficacité des groupes conçus par la stratégie adaptative comparée aux stratégies statiques. Les résultats obtenus lors des différentes expériences effectuées ont permis de justifier l'intérêt de sélectionner des services à base du Log.

6.2 Perspectives

Dans l'objectif d'améliorer notre approche nous proposons différentes perspectives :

1. **Détermination dynamique de la stratégie de tolérance aux fautes (recouvrement)**. Dans le cadre de notre thèse. Nous avons mis en œuvre un certain nombre de mécanismes de tolérance aux fautes, nous avons modélisé les protocoles d'invocation parallèle et séquentiel des services Web divers. Cependant, l'élaboration d'une stratégie complète de recouvrement de services Web, en dépit de fautes, utilisant la diversité requiert la définition des différents détails concernant le groupe de diversité. Dans le but d'expliciter la configuration de groupe de diversité, nous avons traité les problèmes liés à la détermination du nombre nécessaire de services Web divers, ainsi que la détermination des membres de services Web à impliquer dans le groupe. L'approche proposée dans cette thèse peut être étendue par l'identification de la stratégie de tolérance aux fautes appropriée pour une configuration donnée. En réalité, sous certains critères et conditions, un client peut choisir explicitement la stratégie à utiliser. Dans ce cadre, nous avons récapitulé dans quel cas une stratégie doit être choisie et sous quelles conditions. Cependant, l'efficacité de cette proposition est limitée par la forme simple dans laquelle les informations

liées aux types de fautes sont connues, ainsi que les exigences clients en termes des propriétés de *QoS*. Cela pourrait être amélioré à l'avenir par l'élaboration d'un mécanisme de sélection dynamique de la stratégie de tolérance aux fautes (recouvrement) avec un cadre de données plus complet sur le type de fautes, l'origine de la faute, ainsi que les paramètres de *QoS*.

2. **Exploitation de la notion de motifs et items séquentiels pour améliorer la sélection des membres pertinents dans le groupe de diversité**. Dans ce cadre nous avons considéré le problème d'ordonnancement des services Web sélectionnés comme étant un problème de détection des différents itemsets fréquents. Il s'agit après de calculer les différentes fonctions définies dans le chapitre 4 (page 78) pour ordonnancer les services Web réservés. Cet ordonnancement peut être amélioré à l'étape de détection des itemsets fréquents. Plusieurs algorithmes de recherche d'un itemset fréquent existent et peuvent donc être utilisés [Goe05]. Ainsi par exemple, l'algorithme nommé Apriori [AMS+96] (présenté dans le chapitre 4) améliore la recherche d'itemsets fréquents par l'élimination des itemsets qui ne peuvent pas être fréquents parce qu'ils ont un sous ensemble non fréquents d'items.

3. **Médiation sémantique dans un groupe de diversité**. Le déploiement de notre approche nécessitera l'intégration de mécanismes de médiation sémantique au sein d'un groupe de diversité. En effet, le remplacement d'un service concret par un autre garantit la continuité du service à condition que les résultats des services soient compatibles entre eux. Les approches proposées pour les besoins de l'interopérabilité des bases de données et pour l'interopérabilité des services peuvent être intégrés dans le fonctionnement d'un groupe de diversité.

4. **Découverte de services Web sémantiquement équivalents via les réseaux sociaux.** Quoique largement étudiée, la découverte de services Web reste une tâche délicate et longue pour les utilisateurs, qui est due principalement à une offre abondante de services. Les services Web ont longtemps été conçus comme des composants passifs, réagissant uniquement à la demande des utilisateurs. Récemment, une nouvelle tendance serait plutôt de les considérer comme des éléments sociaux qui, dans leur fonctionnement, intégreraient des informations sur leur environnement et leurs interactions passées. Cette dimension sociale découle directement de ces interactions pouvant être assimilées à certains aspects de notre vie quotidienne, par exemple la collaboration ou encore la substitution. En effet, les services Web une fois découverts peuvent se retrouver dans plusieurs situations

d'interaction comme par exemple une composition ou encore un groupe de services assurant la haute-disponibilité d'autres pairs. L'idée est alors d'identifier les différents types d'interactions pouvant exister entre ces services et de représenter ces interactions passées dans des réseaux sociaux afin d'améliorer la qualité de la découverte. Le résultat de cette découverte permettra de construire des groupes de diversité efficaces pouvant s'adapter dans un environnement dynamique.

Bibliographie

[ACKL87] A. Avižienis, W.C. Carter, H. Kopetz, and J.C. Laprie. *The Evolution of fault-tolerant computing*. Dependable computing and fault-tolerant systems. Springer-Verlag, 1987. :1987

[AFGY02] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 429–435, New York, NY, USA, 2002. ACM. :2002

[AFMB12] Hanane Abdeldjelil, Noura Faci, Zakaria Maamar, and Djamal Benslimane. A diversity-based approach for managing faults in web services. In *AINA*, pages 81–88, 2012. :2012

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1) :11–33, January 2004. :2004

[AMS+96] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In U. Fayyad and et al, editors, *Advances in Knowledge Discovery and Data Mining*. MIT Press, 1996. :1996

[AoNuTCS07] T. Anderson and University of Newcastle upon Tyne. Computing Science. *The ReSIST Resilience Knowledge Base*. Technical report series. University of Newcastle upon Tyne, Computing Science, 2007. :2007

[AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *ICDE*, pages 3–14, 1995. :1995

[Avi85] Algirdas Avizienis. The n-version approach to fault-tolerant software. *IEEE Trans. Software Eng.*, 11(12) :1491–1501, 1985. :1985

[Avi95] Algirdas A. Avizienis. *Software Fault Tolerance*, volume 2, chapter The Methodology of N-Version Programming, pages 22–45. John Wiley & Sons, 1995. :1995

[Axi] Axis. http ://ws.apache.org/axis/. 100

[BCP+] S. Bonomi, V. Colaianni, F. Patrizi, D. Pozzi, R. Russo, and M. Mecella. Swsce – an automatic semantic web service composition engine. 10

[Bes10] Xavier Besseron. *Tolérance aux fautes et reconfiguration dynamique pour les applications distribuées à grande échelle*. These, Institut National Polytechnique de Grenoble - INPG, apr 2010. :2010

[BFB11] Jonas Buys, Vincenzo De Florio, and Chris Blondia. Towards context-aware adaptive fault tolerance in soa applications. In *DEBS*, pages 63–74, 2011. :2011

[BLGC10] Diana Borrego, María Teresa Gómez López, Rafael M. Gasca, and Rafael Ceballos. Improving the diagnosability of business process management systems using test points. In *Business Process Management Workshops*, pages 194–200, 2010. :2010

[BS] Zakaria Maamar ÎŠ, Mohammed Lahkim ÎŚ, Djamal Benslimane, and Subramanian Sattanathan. Web services communities- concepts operations-. 86

[BWM07] Stefan Bruning, Stephan Weissleder, and Miroslaw Malek. A fault taxonomy for service-oriented architecture. In *Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium*, HASE '07, pages 367–368, Washington, DC, USA, 2007. IEEE Computer Society. :2007

[CBS+09] K. S. Chan, Judith Bishop, Johan Steyn, Luciano Baresi, and Sam Guinea. Service-oriented computing - icsoc 2007 workshops. chapter A Fault Taxonomy for Web Service Composition, pages 363–375. Springer-Verlag, Berlin, Heidelberg, 2009. :2009

[Cha02] J.M. Chauvet. *Services Web avec SOAP, WDSL, UDDI, ebXML...* Solutions d'entreprise. Eyrolles, 2002. :2002

[Cha04] David A. Chappell. *Enterprise service bus - theory in practice*. O'Reilly, 2004. :2004

[CLK12] Du Wan Cheun, Hyun Jung La, and Soo Dong Kim. A taxonomic framework for autonomous service management in service-oriented architecture. *Journal of Zhejiang University - Science C*, 13(5) :339–354, 2012. :2012

[CLM06] Pat Pik-Wah Chan, Michael R. Lyu, and Miroslaw Malek. Making services fault tolerant. In *ISAS*, pages 43–61, 2006. :2006

[Com] Glassfish Community. https ://jax-ws.dev.java.net/. 101

[Con] World Wide Web Consortium. Hypertext transfer protocol (http). 9

[CR08] Yuhui Chen and Alexander Romanovsky. Improving the dependability of web services integration. *IT Professional*, 10(3) :29–35, 2008. :2008

[CS01] Fabio Casati and Ming-Chien Shan. Models and languages for describing and discovering e-services. In *SIGMOD Conference*, page 626, 2001. :2001

[DF09] V. De Florio. *Application-layer Fault-tolerance Protocols*. Premier reference source. Information Science Reference, 2009. :2009

[DGR04] N. Delgado, A. Q. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Trans. Softw. Eng.*, 30(12), 2004. 16

[DHS05] Glen Dobson, Stephen Hall, and Ian Sommerville. A container-based approach to fault tolerance in service-oriented architectures, 2005. :2005

[DJ07] S. Dustdar and L. Juszczyk. Dynamic replication and synchronization of web services for high availability in mobile ad-hoc networks. *Service Oriented Computing and Applications*, 1(1), 2007. 20

[DK10] Rébecca Deneckère and Elena Kornyshova. La variabilité due à la sensibilité au contexte dans les processus téléologiques. In *INFORSID*, pages 161–176, 2010. :2010

[Erl07] Thomas Erl. *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007. :2007

[ESLH08] Christian Engelmann, Stephen L. Scott, Chokchai Leangsuksun, and Xubin He. Symmetric active/active high availability for high-performance computing system services : Accomplishments and limitations. In *CCGRID*, pages 813–818, 2008. :2008

[FAMB11] Noura Faci, Hanane Abdeldjelil, Zakaria Maamar, and Djamal Benslimane. Using diversity to design and deploy fault tolerant web services. In *WETICE*, pages 73–78, 2011. :2011

[FB02] Dieter Fensel and Christoph Bussler. Semantic web enabled web services. *Lecture notes in computer*, 2342 :1–2, 2002. :2002

[FCMJ12] R. Z. Frantz, R. Corchuelo, and C. Molina-Jiménez. A proposal to detect errors in enterprise application integration solutions. *Journal of System and Software*, 85(3), 2012. 18

[Fie] Roy Thomas Fielding. http ://www.ics.uci.edu/ fielding/pubs/dissertation/rest-arch-style.htm. 9

[FKP10] Jean-Charles Fabre, Marc-Olivier Killijian, and Thomas Pareaud. Towards on-line adaptation of fault tolerance mechanisms. In *EDCC*, pages 45–54, 2010. :2010

[Flo09] Vincenzo De Florio. Software assumptions failure tolerance : Role, strategies, and visions. In *WADS*, pages 249–272, 2009. :2009

[GBYMN07] Ghada GASMI, S. Ben Yahia, and Engelbert Mephu Nguifo. Extraction de règles d'association à partir de données décentralisées : Problèmes et perspectives. In Boussaid O. et Masseglia F. (eds.), editor, *Atelier Fouille*

de Données Complexes (FDC'07) de la conf. EGC'07, pages 75–80, Namur, Belgique, jan 2007. :2007

[GKR09] Anatoliy Gorbenko, Vyacheslav S. Kharchenko, and Alexander Romanovsky. Using inherent service redundancy and diversity to ensure web services dependability. In *Methods, Models and Tools for Fault Tolerance*, pages 324–341. 2009. :2009

[Goe05] Bart Goethals. Frequent set mining. In Oded Maimon and Lior Rokach, editors, *The Data Mining and Knowledge Discovery Handbook*, pages 377–397. Springer, 2005. :2005

[HB04] Hugo Haas and Allen Brown. Web Services Glossary - W3C Working Group Note. Technical report, World Wide Web Consortium - W3C, 11 February 2004. :2004

[HPMA+00] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Freespan : frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pages 355–359, New York, NY, USA, 2000. ACM. :2000

[HSS+04] Jason O. Hallstrom, Nigamanth Sridhar, Paolo A. G. Sivilotti, Anish Arora, and William M. Leal. A container-based approach to object-oriented product lines. *JOURNAL OF OBJECT TECHNOLOGY*, 2004. :2004

[HXW+07] Yu Huang, Chunxiang Xu, Hanpin Wang, Yunni Xia, Jiaqi Zhu, and Cheng Zhu. Formalizing web service choreography interface. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops - Volume 02*, AINAW '07, pages 576–581, Washington, DC, USA, 2007. IEEE Computer Society. :2007

[Jur06] Matjaz B. Juric. *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition*. Packt Publishing, 2006. :2006

[KL86] John C. Knight and Nancy G. Leveson. An experimental evaluation of the assumption of i ndependence in multi-version programming *. *IEEE Transactions on Software Engineering*, 12 :96–109, 1986. :1986

[KSK] Markus Keidl, Stefan Seltzsam, and Alfons Kemper. Reliable web service execution and deployment in dynamic environments. In *In Proceedings of the International Workshop on Technologies for E-Services (TES*, pages 104–118. 24

[KSK03] Markus Keidl, Stefan Seltzsam, and Alfons Kemper. Reliable web service execution and deployment in dynamic environments. In *TES*, pages 104–118, 2003. :2003

[Lab] JBoss Labs. http ://labs.jboss.com/jbossws/. 100

[Lap04] Jean-Claude Laprie. Dependable computing : Concepts, challenges, directions. In *COMPSAC*, page 242, 2004. :2004

[LCR06] Peter Li, Yuhui Chen, and Alexander Romanovsky. Measuring the dependability of web services for use in e-science experiments. In *Proceedings of the Third international conference on Service Availability*, ISAS'06, pages 193–205, Berlin, Heidelberg, 2006. Springer-Verlag. :2006

[LHS08] Lingxi Li, Christoforos N. Hadjicostis, and R. S. Sreenivas. Designs of bisimilar petri net controllers with fault tolerance capabilities. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 38(1) :207–217, 2008. :2008

[LL04] Ming-Yen Lin and Suh-Yin Lee. Incremental update on sequential patterns in large databases by implicit merging and efficient counting. *Inf. Syst.*, 29(5) :385–404, 2004. :2004

[LMX05] Nik Looker, Malcolm Munro, and Jie Xu. Increasing web service dependability through consensus voting. In *COMPSAC (2)*, pages 66–69, 2005. :2005

[LRD11] Christine Louberry, Philippe Roose, and Marc Dalmau. Kalimucho : Contextual deployment for qos management. In *DAIS*, pages 43–56, 2011. :2011

[MCP98] F. Masseglia, F. Cathala, and P. Poncelet. The psp approach for mining sequential patterns. pages 176–184, 1998. :1998

[Mica] Sun Microsystems. https ://java.net/projects/jax-rpc/. 101

[Micb] Sun Microsystems. http ://www.oracle.com/technetwork/java/index.html. 100

[MPT03] Florent Masseglia, Pascal Poncelet, and Maguelonne Teisseire. Incremental mining of sequential patterns in large databases. *Data Knowl. Eng.*, 46(1) :97–121, 2003. :2003

[MSR77] P. M. Melliar-Smith and Brian Randell. Software reliability : The role of programmed exception handling. In *Language Design for Reliable Software*, pages 95–100, 1977. :1977

[Net] NetBeans. http ://www.netbeans.org/kb/. 101

[OAS04] *Universal Description, Discovery, and Integration (UDDI) - Version 3*, October,2004. UDDI Spec Technical Committee Draft. :2004

[Ora] Oracle. https ://glassfish.dev.java.net/javaee5/docs/docsindex.html. 100

[Org07] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*, April 2007. 17

[PKPS02] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic matching of web services capabilities. In *International Semantic Web Conference*, pages 333–347, 2002. :2002

[PTDL03] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing. *Communications of the ACM*, 46, 2003. 17

[PYB+99] Nicolas Pasquier, Yves, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24 :25–46, 1999. :1999

[Ran75] Brian Randell. System structure for software fault tolerance. *IEEE Trans. Software Eng.*, 1(2) :221–232, 1975. :1975

[RCS01] Jennifer Ren, Michel Cukier, and William H. Sanders. An adaptive algorithm for tolerating value faults and crash failures. *IEEE Trans. Parallel Distrib. Syst.*, 12(2) :173–192, 2001. :2001

[RRS+97] Alexander Romanovsky, Brian Randell, Robert J. Stroud, Jie Xu, and Avelino Francisco Zorzo. Implementation of blocking coordinated atomic actions based on forward error recovery. *Journal of Systems Architecture*, 43(10) :687–699, 1997. :1997

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns : Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology : Advances in Database Technology*, EDBT '96, pages 3–17, London, UK, UK, 1996. Springer-Verlag. :1996

[Saa01] T.L. Saaty. *Decision Making with Dependence and Feedback : The Analytic Network Process : the Organization and Prioritization of Complexity*. Analytic hierarchy process series. Rws Publications, 2001. :2001

[SF07] Nicolas Salatge and Jean-Charles Fabre. Fault tolerance connectors for unreliable web services. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN'07, pages 51–60, Washington, DC, USA, 2007. IEEE Computer Society. :2007

[SHD05] Ian Sommerville, Stephen Hall, and Glen Dobson. Dependable service engineering ; a fault-tolerance based approach dependable service engineering : A fault-tolerance based abstract approach, 2005. :2005

[SOA02] *SOAP Version 1.2 Part 0 : Primer*, 2002. Working Draft. :2002

[SPJ11] Sebastian Stein, Terry R. Payne, and Nicholas R. Jennings. Robust execution of service workflows using redundancy and advance reservations. *IEEE Trans. Serv. Comput.*, 4(2) :125–139, April 2011. :2011

[TCZH05] W-T. Tsai, Y. Chen, D. Zhang, and H. Huang. Voting multi-dimensional data with deviations for web services under group testing. In *International Conference on Distributed Computing Systems Workshops*, 2005. :2005

[TIRL03] F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy. Coordinated forward error recovery for composite web services. In *In Symposium on Reliable Distributed Systems (SRDS*, 2003. 20

[toE00] toExcel. *Extensible Markup Language (Xml) 1.0 Specifications : From the W3c Recommendations*. iUniverse, Incorporated, 2000. :2000

[Top03] Kim Topley. *Java Web Services in a Nutshell*. O'Reilly, Beijing, 2003. :2003

[WSWW06] F. Wagner, R. Schmuki, T. Wagner, and P. Wolstenholme. *Modeling Software with Finite State Machines : A Practical Approach*. Modeling Software with Finite State Machines : A Practical Approach. Taylor & Francis, 2006. :2006

[WZV10] Yao Wang, Jie Zhang, and Julita Vassileva. Effective web service selection via communities formed by super-agents. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '10, pages 549–556, Washington, DC, USA, 2010. IEEE Computer Society. :2010

[YD07] Yuhong Yan and Philippe Dague. Modeling and diagnosing orchestratedweb service processes. In *ICWS*, pages 51–59, 2007. :2007

[Zak01] Mohammed J. Zaki. Spade : An efficient algorithm for mining frequent sequences. In *Machine Learning*, pages 31–60, 2001. :2001

[ZD05] Albert Y. Zomaya and Hassan B. Diab. *Dependable Computing Systems : Paradigms, Performance Issues, and Applications (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2005. :2005

[ZL10] Zibin Zheng and Michael R. Lyu. An adaptive qos-aware fault tolerance strategy for web services. *Empirical Software Engineering*, 15(4) :323–345, 2010. :2010

Publications

Conférences internationales avec comité de lecture

1. H.Abdeldjelil, N.Faci, Z.Maamar, D.Benslimane, Diversity-Based Approach For Managing Faults in Web Services. Dans IEEE International Conference on Advanced Information Networking and Applications (AINA), Fukuoka, Japan. 2012.

2. N.Faci, H.Abdeldjelil, Z.Maamar, D.Benslimane, Using Diversity to Design and Deploy Fault Tolerant Web Services. Dans International Conference on Collaboration Technologies and Infrastructures (WETICE), Paris, France.IEEEXplore . 2011.

3. N.Faci, H.Abdeldjelil, Z.Maamar, D.Benslimane, Vers un framework intégrant les principes des réseaux sociaux dans la découverte de services Web. Dans Conférence Internationale sur les NOuvelles Technologies de la REpartition (NOTERE'11), Paris, France. IEEE Xplore. 2011.

Fault Tolerant (FT) Web services are components with higher resilience to failures that result out of various unexpected faults for instance software bugs and machine crashes. Since it is impractical to predict the potential occurrence of a fault, a widely used strategy consists of duplicating, in a passive or active way, critical components (e.g., Web services) that interact during a distributed application execution (e.g., composition). The ability of this application to continue operation despite component failures is referred to as Fault Tolerance (FT). Duplication is usually put forward as a solution to make these components fault tolerant. It is achieved through either replication or diversity. In this thesis, we are particularly interested in diversity, and we show how semantically similar Web services, i.e., offer same functionality (e.g., Weather Forecast) but implement this functionality differently in terms of business logic and technical resources, collaborate together to make web services fault tolerant. We illustrate the limitations of replication (e.g., presence of replicated faults) and suggests diversity as an alternative solution. Our literature review revealed a limited interest in diversity for FT Web services.

We first discuss the value-added of diversity to support fault tolerant Web Services operation. Semantically similar Web services are gathering into the same virtual space referred to as diversity group. Building upon this similarity, a diversity group consisting of semantically similar Web services is built and then controlled using different execution models. Each model defines how the Web services in the diversity group collaborate and step in when one of them fails to ensure operation continuity. Besides the diversity group additional components are used including : composite manager, group manager, and Web service manager. The first component invokes the diversity groups and either resumes or aborts the overall execution in case of failures. The second component oversees Web services execution and the interactions between them. Finally, the last component monitors a Web service execution and reports either failure or success to the group manager.

Setting up an appropriate diversity strategy also requires working out different details such as the necessary number of semantically Web services, location of these Web services, and protocols that coordinate the functioning of these Web services. In the existing diversity approaches, all available Web services in a diversity group are put into action. However this might not be doable all the time ; Web services performance requires resources that can be limited and sometimes costly. Moreover, as the QoS values (e.g., availability and performance) required to the group can vary from one composition to another, only a subset of the Web services would be necessary. So, different configurations are offered taking into account factors like availability degree and functional deviation to accept. In this work, we raise questions related to how many similar Web services are necessary to form a diversity group, when and how a similar Web service needs to step in to take over from a failing Web service, and how to oversee the operation of all these Web services without violating non-functional requirements such as privacy.

We develop an adaptive diversity- based framework to make Web services FT that supports design and deployment of reliable composite Web services. This framework dynamically adapts the composite Web service by continuously monitoring the behavior of the component Web services in the diversity groups in order to react appropriately to faults occurring at runtime. It also offers several services like discovery/selection, failure detection, and recovery. Moreover, it provides developers with guidelines for designing and deploying FT Web services. To validate our approach, we run a set of experiments showing the use of diversity and its efficiency.

Keywords : Dependability, fault tolerant, Web service, selection