République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université Badji Mokhtar -Annaba-

Faculté des Sciences de l'Ingéniorat

Département d'Informatique

THÈSE

Présentée en vue de l'obtention du diplôme de

Doctorat en Sciences

Spécialité : Informatique

Présentée par :

KHEBIZI ALI

Vers une approche déclarative pour l'analyse de l'impact de l'évolution dynamique des protocoles des services web

Soutenue publiquement
à Annaba le 11 Juillet 2017
Devant le jury composé de :

JURY

Pr. Djamel Meslati	Université Badji Mokhtar, Annaba	Président
Pr. Hassina Séridi-Bouchelaghem	Université Badji Mokhtar, Annaba	Directrice de thèse
Pr. Farouk Toumani	Université Blaise Pascal, Clermont Ferrand France	Co-directeur
Pr. Allaoua Chaoui	Université Abdelhamid Mehri, Constantine 2	Examinateur
Pr. Abdelkrim Amirat	Université Chérif Messaadia, Souk Ahras	Examinateur
Pr. Farid Mokhati	Université Larbi Ben M'Hidi, Oum El Bouaghi	Examinateur

« *Rien n'est permanent, sauf le changement.* »

- Héraclite d'Ephèse

DEDICACES

A la mémoire de ma mère et de mes frères

A mon père pour sa résistance et ses sacrifices

A ma femme Halima et mes enfants : Raji, Rami, Anis et Nardjes

REMERCIEMENTS

J'exprime mes profonds remerciements, à ma directrice de thèse : Madame **Séridi-Bouchelaghem Hassina**, Professeur à l'université Badji Mokhtar- Annaba- pour son soutien infaillible, sa disponibilité permanente et ses encouragements continus. Je la remercie pour le temps qu'elle a consacré à la relecture de cette thèse et à la révision des articles publiés, ainsi que pour ses orientations et remarques qui ont fait aboutir cette thèse.

Mes plus vifs remerciements à mon co-directeur Mr : **Toumani Farouk**, Professeur à l'université Blaise Pascal-Clermont Ferrand -France- pour son accueil au niveau de l'équipe Données-Services et Interopérabilité du laboratoire LIMOS. Les séjours passés dans ce laboratoire m'ont été très bénéfiques et enrichissants aussi bien sur le plan scientifique qu'humain. Je le remercie aussi pour m'avoir fait découvrir le goût de la recherche. Ses conseils ont permis de donner une orientation plus formelle à la présente thèse, chose que j'apprécie beaucoup.

Je tiens également à remercier les membres de jury qui m'ont fait l'honneur de bien vouloir évaluer mon travail, et plus précisément :

- Monsieur : **Meslati Djamel**, Professeur à l'Université Badji Mokhtar d'Annaba pour l'honneur qu'il m'a fait, en acceptant de présider ce jury.
- Monsieur : **Chaoui Allaoua**, Professeur à l'Université Abdelhamid Mehri, Constantine 2 pour avoir accepter de lire et d'examiner le présente thèse.
- Monsieur : **Mokhati Farid**, Professeur à l'Université Larbi Ben M'Hidi, Oum El Bouaghi pour avoir accepter d'évaluer et d'examiner ce manuscrit.
- Monsieur : **Abdelkrim Amirat**, professeur à l'Université Chérif Messaadia, Souk Ahras d'avoir bien voulu se pencher sur ce travail et faire partie de ce jury.

Mes sincères remerciements aux Professeurs : **Bouallem Benatallah** de l'Université New South Wales (**UNSW**) de Sidney -Australie- et **Houari Nourine** de l'université Blaise Pascal de Clermont Ferrand -France- pour les riches discussions scientifiques qui ont fait progresser cette thèse.

Enfin, je remercie vivement ma grande famille de m'avoir encouragé à réaliser cette thèse et pour son soutien au quotidien.

RESUME

Les services web constituent la nouvelle génération de composants logiciels favorisant l'intégration des systèmes d'informations hétérogènes et distribués. Dans de tels environnements, les organisations implémentent leurs processus métiers en tant que services web. Par ailleurs, pour adapter leurs systèmes d'information à la versatilité de l'environnement, les organisations investissent des budgets et des efforts colossaux dans la gestion de l'évolution des processus métiers. Cependant, l'un des problèmes majeurs auquel est confrontée la technologie des services web est le développement de techniques et d'outils pour la prise en charge des changements qui affectent la description des services qui sont déjà déployés et qui sont invoqués par divers clients. En effet, dans le contexte des évolutions dynamiques, la continuité des instances actives, conformément au nouveau processus est un problème crucial qui exige une analyse de l'impact des changements des processus métiers sur les instances actives d'un service.

Dans cette thèse, nous abordons le problème de l'évolution dynamique des services web et nous analysons l'impact des changements de leurs protocoles sur les instances en cours d'exécution.

Alors que les travaux antérieurs ayant traité ce problème se sont concentrés, essentiellement, sur la vérification de certains critères de conformité ou sur le calcul des régions du schéma du protocole qui sont affectées par les changements afin d'assurer la migration des instances actives d'une manière simpliste (*migrables, non migrables*), nous proposons une approche déclarative permettant d'assurer différentes stratégies de migration des instances actives vers la nouvelle version du protocole.

Le trait saillant de notre approche consiste en un cadre formel fondé sur un langage déclaratif pour soutenir les fournisseurs de services à définir leurs propres stratégies de migration, en leur donnant la possibilité de spécifier d'une manière déclarative les contraintes qui gouvernent le processus de migration des instances, au lieu de les forcer à refléter le plus fidèlement possible celles d'origine. Nous percevons la migration d'instances comme étant le problème de prendre des instances s'exécutant sous un protocole donné et de les convertir en de nouvelles instances dans le nouveau protocole.

Le cadre formel proposé est fondé sur la spécification d'un langage de migration d'instances, et sur l'identification et la formalisation d'un ensembles de patrons de migration. L'approche proposée permet aux fournisseurs de services de spécifier leur propres stratégies de migration, suite à l'évolution d'un protocole de service. Les stratégies de migration sont exprimées par un ensemble de patrons de migration, et chaque patron capture un classe de contrainte et peut être instancié par le fournisseur de service pour définir des règles de migration spécifiques dans le cadre d'une stratégie de migration particulière. Les patterns proposés peuvent êtres composés en utilisant les opérateurs logiques pour définir des stratégies de migration plus élaborées.

L'approche proposée a été implémentée, expérimentée et évaluée. L'outil logiciel réalisé offre des fonctionnalités très utiles aux gestionnaires des protocoles métier pour la gestion de la migration des instances actives des services web.

Mots-clés : *Gestion des changements, protocole de service, évolution de protocole, instance active, chemin d'exécution, expression de chemin, patron de migration.*

ABSTRACT

Web services are the new generation of software components enabling the integration of heterogeneous and distributed information systems. One of the major issues that must be handled by web services infrastructure consists to the development of techniques and tools for managing changes in service descriptions and for analysing the impact of changes on active instances of a service which was already published and invoked by customers. Indeed, to adapt their information systems to the environment versatility, organizations are investing tremendous efforts and budgets for the maintenance and the management of changes that affect their business processes. However, in the context of dynamic evolution, the continuity of the active instances, according to the new business process is a challenging issue.

In this thesis, we address the problem of the dynamic evolution of web services and we analyse the impact of their changes on running instances.

While the existing approaches for instances migration has focussed mainly on the verification of certain compliance criteria or on computing the region of the protocol scheme that are affected by changes in order to ensure the migration of active instance in a simple black and white fashion (migrateable, not migrateable), while forcing the migrated instances to reflect the original instances as accurately as possible, in our approach we give to service providers the ability to declaratively define the constraints that drive the instances migration process.

To this end, we perceive instances migration as the problem of taking instances running under a given protocol and to **convert** them into new instances of the new protocol.

The salient feature of the proposed framework lies in a declarative language to support service providers in defining fine-grained instances migration strategies of active instances, by giving them the ability to specify in a declarative manner the set of constraints that govern the migration process of active instances, instead of forcing them to reflect as closely as possible the original ones.

We propose a formal framework based on an instances migration language and on the identification and the formalization of a set of customizable migration patterns which are intended to be used to facilitate the specification of migration strategies. A migration pattern is a high level abstraction which captures a class of application dependent constraints and can be instantiated by a service provider to define a context-specific migration rule that maps old instances to new ones. The proposed patterns can be composed using logical operators (*e.g., conjunction, disjunction*) to define powerful migration strategies.

The proposed approach has been implemented, experimented and its completeness was evaluated. The developed software tool offers useful features to business protocol managers for handling the migration of running instances of web services.

Keywords : *Change management, service protocol, protocol evolution, active instance, execution path, path expression, migration pattern.*

Table des matières

Table des figures

Liste des tableaux

Liste des symboles

<**AFD**> <Automate Fini Déterministe>

<**API**> <Application Programming Interface>

<**BAM**> <Business Activity Monitoring>

<**BDD**> <Base De Données>

<**BPEL**> < Business Process Execution Language>

<**BPE**> <Business Process Engine>

<**BPML**> <Business Process Modeling Language>

<**BPMN**> <Business Process Modeling Notation>

<**BPMS**> <Business Process Management System>

<**BPM**> <Business Process Management>

<**BPR**> <Business Process Reengineering>

<**BP**> <Business Process>

<**CGI**> <Commun Gateway interface>

<**CORBA**> <Common Object Request Broker Architectur>

<**DBMS**> <Database Management System>

<**DCOM**> <Distributed Component Object Model>

<**DLL**> <Dynamic Link Library>

<**DTD**> <Document Type Definition>

<**FTP**> <File Transfer Protocol>

<**GUI**> <Graphic User Interface>

<**HTML**> <HyperText Markup Language>

<**HTTP**> <HyperText Transfer Protocol>

<**IDE**> <Integrated Development Environment>

<**IDL**> <Interface Desciption Language>

<**JDOM**> <Java Document Object Model>

<**JMS**> <Java Messaging System>

<**JRMI**> <Java Remote Method Invocation>

<**MCT**> <Modèle Conceptuel des Traitements>

<**MIME**> <Multipurpose Internet Mail Extensions>

<**OASIS**> <Advancing Open Standards for the Information Society>

<**OMG**> <Object Management Group>

<**OSGi**> <Open Services Gateway initiative>

<**OWLS**> <Ontology Web Language for Service>

<**PAIS**> <Process Aware Information System>

<**PCIA**> <Protocol Change Impact Analyser>

<**QoS**> <Quality of Service>

<**RdP**> <Réseaux de Petri>

<**RPC**> <Remote Procedure Call>

<**SGBDR**> <Système de Gestion de Base de Données Relationnel>

<**SGBD**> <Système de Gestion de Base de Données>

<**SGML**> <Standard Generalized Markup Language>

<**SI**> <Système d'informations>

<**SMTP**> <Simple Mail Transfer Protocol>

<**SOAP**> <Simple Object Access protocol>

<**SOA**> <Service Oriented Architecture>

<**SQL**> <Structured Query Language>

<**UDDI**> <Universal Description, Discovery and Integration>

<**UML**> <Unified Modeling Language>

<**URI**> <Uniform Ressource Identifier>

<**URL**> <Uniform Ressource Locator>

<**W3C**> <World Wide Web Consortium>

<**WfMC**> <Workflow Management Coalition>

<**WfMS**> <Workflow Management System>

<**WSCI**> <Web Services Choreography Interface>

<**WSCL**> <Web Service Conversation Language>

<**WSDL**> <Web Services Description Language>

<**WSFL**> <Web Service Flow Language>

<**xCBL**> <XML Common Business Library>

<**XKMS**> <XML Key Management Specification>

<**XLANG**> <Xml LANGuage>

<**XML**> <eXtensible Markup Language>

<**XSLT**> <eXtensible Stylesheet Language Transformations>

<**XSL**> <eXtensible Stylesheet Language>

Chapitre 1

Introduction générale

L'introduction générale de cette thèse présente d'abord son contexte global dans le domaine des services web et de l'évolution des processus métiers. Puis, la problématique de recherche abordée dans ce travail de recherche est exposée, illustrée et motivée. Elle est suivie des objectifs visés et des contributions majeures apportées dans la thèse. Enfin, un aperçu de la structure du manuscrit est présenté.

1.1 Le contexte de la thèse

Le phénomène de la globalisation de l'économie, conjugué aux progrès réalisés dans le domaine des technologies de l'information et de la communication, ont bouleversé complètement les modes de fonctionnement des organisations modernes. En effet, aujourd'hui, les entreprises économiques et les administrations sont assujetties à des facteurs environnementaux en perpétuelle évolution. Ces facteurs les contraignent, impérativement, à accroître leurs capacités de réaction afin de s'adapter aux changements qui sont de plus en plus fréquents et qui se produisent avec un rythme très accéléré. Par ailleurs, la dimension trans-frontalière et la montée en puissance de nouvelles formes de coalitions économiques, obligent les entreprises modernes à affronter des conditions concurrentielles très rudes.

La conséquence directe de ces constats est que la survie et la pérennité de l'entreprise dépendent étroitement de sa capacité d'adaptation aux changements qui affectent ses micro et macro environnements.

Pour faire face à ces enjeux stratégiques et afin de préserver ses parts du marché mondial, l'entreprise doit être versatile, flexible et collaborative, tout en demeurant réactive pour prendre en charge les facteurs qui surgissent dans son écosystème. Cette réactivité passe, inévitablement, par la maitrise des processus métiers gérés par l'organisation et par leur adaptation continue aux conditions mouvantes de l'environnement. En effet, pour adapter leurs systèmes d'informations aux mutations de l'environnement, les organisations investissent des budgets et des efforts colossaux pour la maintenance et la gestion de l'évolution de leurs processus métiers. Cependant, dans le contexte des évolutions dynamiques (*i.e : lorsque les changements sont opérés sur des processus métiers alors que des instances des processus en question sont en cours d'exécution*), la continuité des instances actives ayant démarré leur interactions sur la base d'une ancienne version du processus métier, est un problème crucial.

D'autre part, différents concepts, architectures, méthodes et technologies, tels que : les architectures orientées services [4, 5], la gestion des processus métiers (**BPM**[1]) [3] et les services web [1] ont émergé durant la dernière décennie, afin de soutenir les besoins croissants des organisations pour l'adaptation rapide et efficace aux conditions

1. **B**usiness **P**rocess **M**anagement

très variables de leur environnement. Dans ce nouveau paysage, la technologie des services web constitue, aujourd'hui, une solution de choix pour l'intégration des systèmes d'information intra et inter-entreprises permettant de surmonter les problèmes d'interopérabilité et d'hétérogénéité des systèmes d'information des organisations. Dans de tels environnements, les processus métiers sont implémentés en tant qu'applications qui sont publiées sur Internet comme des services web. Ces applications peuvent ainsi être découvertes et invoquées par différents partenaires via des protocoles standards (WSDL [6], SOAP [7], UDDI [8]). Par ailleurs, les services web offrent des techniques de composition favorisant la création de nouvelles applications informatiques à plus forte valeur ajoutée, et ce par articulation de celles qui existent déjà [9].

Dans un tel contexte caractérisé par l'ouverture, l'universalité et la réactivité, la gestion de l'évolution d'un service web constitue un aspect très important de son cycle de vie et se pose, alors la question de la continuité des instances actives, suite à l'application des changements sur la description des services web. Cette question est, particulièrement, pertinente quand les changements sont opérés alors que des instances du service sont en cours d'exécution au moment du changement.

Dans cette thèse, nous considérons que les processus métiers sont implémentés en tant que services web. Les descriptions des comportements externes de tels services sont spécifiées à l'aide de la notion de **protocole de service** qui permet de décrire le comportement externe et visible du service web [10], en spécifiant un ensemble de contraintes que les clients doivent respecter afin d'interagir correctement avec le service [11]. Les protocoles de service sont représentés par des automates d'états finis déterministes. Ainsi, l'application des changements sur les processus métiers de l'entreprise se traduit par des modifications de la description des protocoles des services web qui les représentent.

1.2 Problématique

Les changements dans les processus métiers peuvent être motivés par des raisons très variées, telles que : les modifications des lois et des réglementations, la mise à niveau des procédures de travail, la maintenance des règles de gestion, les changements dans la structure de l'organisation, l'optimisation des processus métiers, les rachats et fusions d'entreprises, ... etc. Comme conséquence immédiate à ces changements, les entreprises doivent continuellement mettre à jour et améliorer leurs processus métiers qui sont publiés sur le web en tant que services web, afin de s'adapter aux évolutions qui se produisent à un rythme de plus en plus fréquent.

Pour faire face aux défis inhérents aux évolutions des processus métiers, deux approches peuvent être mises en œuvre. La première approche est dite statique, du fait que les changements dans la spécification d'un processus métiers sont effectués alors qu'il n'y a pas d'instances de ce processus en cours d'exécution. Dans ce cas, les changements prendront effet uniquement pour les nouvelles instances et ils n'auront généralement pas d'impact sur les instances déjà terminées. La seconde approche est qualifiée de dynamique, dans le sens où les changements sont opérés sur un processus métier alors que des instances de ce processus sont en cours d'exécution au moment du changement. Dans cette deuxième situation, se pose alors la question cruciale liée à la gestion des instances actives d'un processus métier dont la spécification a été

modifiée. Ce scénario est très fréquent en pratique car les processus métiers réels ont des durées d'exécution relativement longues (*de quelques minutes à quelques jours ou semaines*), et il arrive souvent d'avoir à un instant donné plusieurs milliers d'instances d'un service qui sont actives en même temps. Dès lors, le problème qui se pose est de savoir **comment traiter les instances actives et comment gérer leur migration vers la nouvelle spécification du processus métier ?**.

La figure 1.1, ci-dessous met en exergue la problématique abordée dans cette thèse.

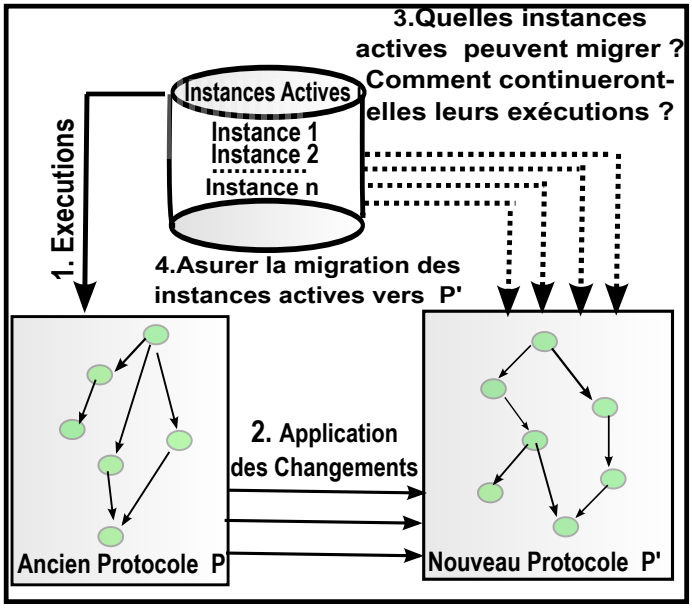

FIGURE 1.1 – Problématique de la migration des instances actives suite aux évolutions

Dans le spectre des solutions possibles au problème de la gestion des évolutions dynamiques des processus métiers, deux options possibles consistent à : (i) soit laisser toutes les instances actives poursuivre leur exécution selon l'ancien processus métier, ou (ii) annuler toutes les instances actives et les redémarrer conformément à la nouvelle spécification du processus métier.

Cependant, ces deux solutions extrêmes sont rarement possibles en pratique et il y a un besoin réel d'analyse de l'impact du changement pour les raisons suivantes :

- **Préservation (*partielle*) de l'historique des exécutions :** dans plusieurs situations, il est impératif que les instances en cours puissent continuer leur exécutions, même si la spécification du protocole a changé. C'est la cas, par exemple, quand certaines activités ne peuvent être annulées et leurs effets ne peuvent être compensés. Dans ce cas, la stratégie de migration doit assurer la préservation, même partielle de l'historique des exécutions. En outre, la préservation de l'historique peut être souhaitable pour des raisons de performance. En effet, du fait que les instances ont déjà effectué des activités, il n'est pas fonctionnel de demander aux clients, qui sont en interaction avec le service, de redémarrer leurs instances de processus à partir de zéro tout en annulant le travail déjà effectué, car cette façon de faire peut être catastrophique pour eux (*perte de travail*).

– **Mise en conformité instantanée avec les règles métiers :** certains changements dans un processus métier, par exemple, en raison de nouvelles lois ou réglementations, doivent entrer en vigueur immédiatement et dans un délai très court. Dans ce cas, il n'y a pas d'autres choix que d'appliquer les changements instantanément et de basculer (*faire migrer*) les instances en cours d'exécution vers la nouvelle version du processus afin de se conformer aux nouvelles exigences.

En plus des deux raisons fondamentales précédentes, la gestion de la migration des instances actives vise à répondre aux motivations suivantes :

– **Assurer la continuité du service :** lors des changements des processus métiers, le bon fonctionnement des applications ne doit pas être entravé. En fait, assurer la continuité de service est une préoccupation majeure des administrateurs des applications d'entreprises, car dans les systèmes temps-réel et dans de nombreuses applications métier (*aéronautique, e-commerce, les systèmes médicaux, systèmes de contrôle, l'industrie, ...*), une interruption même temporaire du service, aux fins de maintenance, peut être lourde de conséquences pour l'organisation. Ainsi, il est impératif de traiter avec précision l'impact des changements sur les instances actives afin de préserver les vies humaines, les biens et aussi l'image de marque des entreprises.
– **Amélioration des performances :** pour de nombreuses applications d'entreprise et les applications grand public (*commerce électronique, e-gouvernement, e-learning ...*), un nombre important d'instances du processus est en cours d'exécution en même temps. La gestion manuelle de la migration constitue, alors, une tâche très lourde pour les gestionnaires des applications. Un outil logiciel permettant l'analyse automatique de l'impact des changements et la gestion transparente de la migration devient plus que indispensable.

Cette thèse vise à répondre aux préoccupations précédentes en abordant les questions suivantes :

1. Que faire en cas de changement du schéma d'un processus métier ? Comment traiter les instances en cours d'exécution ?
2. Comment les instances actives d'un processus métier sont transférées (*migrées*) vers une nouvelle spécification du processus, suite à des modifications de la logique métier ?
3. Quel est l'impact de l'évolution des processus métiers sur les instances actives ?
4. Quels sont les critères pertinents à prendre en compte lors de la migration des instances actives d'un protocole qui a subi des changements ?
5. Le gestionnaire de protocoles peut-il spécifier et/ou choisir ses propres stratégies de migration pour répondre à des besoins opérationnels spécifiques ?

1.3 Objectifs et contributions de la thèse

La littérature consacrée aux problèmes de gestion des changements dans les environnements dynamiques est extrêmement riche. Elle soulève plusieurs problèmes et des difficultés de natures différentes, et la plupart d'entre a été largement étudiée.

Dans le domaine des workflows et des systèmes d'information conscients des processus (*Process Aware Information Systems :* ***PAIS***), plusieurs travaux ont traité

divers aspects du problème de l'évolution des processus métiers ou des workflows [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]. En effet, plusieurs travaux se sont intéressés aux questions relatives à l'évolution des schémas des processus métiers/workflow, e.g., contrôle de versions des processus [13], prise en compte de la propagation des changements [12], spécification des primitives de modification et les propriétés associées [15] et proposition des patrons d'adaptation [14]. Cependant, une majorité des approches proposées s'est concentrée essentiellement sur la vérification de critères de conformité afin d'assurer que la migration d'instances ne souffre pas de problèmes tels que l'*interblocage* ou la *suppression d'une activité déjà exécutée)* (*voir [23] pour un état de l'art plus détaillé*). D'autres travaux, e.g., [18], proposent de calculer les régions du schéma d'un workflow qui sont affectées par les changements. Ces dernières sont ensuite exploitées pour identifier les instances qui peuvent migrer (*i.e., les instances qui ne sont pas dans les régions affectées par les changements*). Dans ce type d'approches, toutes les instances actives sont gérées d'une manière simple (*en noir et blanc*), *migrables* ou *non migrable*, et les besoins de migration spécifiques des diverses instances actives ne sont pas pris en compte.

Plus directement liés à nos préoccupations, certains travaux de recherche dans le domaine des services web [21, 24, 25] ont récemment abordé le problème de la migration transparente des instances actives lors de l'évolution des protocoles métier des services web. En se basant sur les notions de compatibilité historique et compatibilité future, [24] présente un ensemble de méthodes pour identifier automatiquement les instances d'un service qui peuvent migrer vers un nouveau protocole. [21] étend les techniques de [24] pour les cas des protocoles de services asynchrones et non-déterministes.

Le travail présenté dans cette thèse s'inscrit dans cette lignée de travaux de recherche. Il s'appuie sur les travaux existants afin de proposer une nouvelle approche pour la gestion de la migration des instances actives lors de l'évolution des protocoles de services web. Le trait saillant du travail proposé réside dans la spécification d'une approche **déclarative** fondée sur un cadre formel qui permet d'offrir aux fournisseurs de services un support logiciel pour définir des stratégies de migration paramétrables et à **granularité fine**, afin d'assurer la **continuité des exécutions** des instances actives, suite à l'application des changements sur la description des processus métiers.

Pour atteindre cet objectif, nous considérons la migration d'instances comme étant le problème de prendre les instances s'exécutant sous un protocole donné et de les *convertir* en de nouvelles instances dans le nouveau protocole. Tandis que les approches existantes pour la migration forcent les instances actives à faire migrer à refléter, le plus fidèlement possible les instances originales, dans notre approche nous donnons aux fournisseurs de services la possibilité de définir d'une **manière déclarative** les contraintes qui gouvernent le processus de migration des instances actives et de spécifier, par là même, leurs propres **stratégies de migration**.

Dans notre approche, une stratégie de migration est exprimée par un ensemble de **règles de migration** qui sont spécifiées en utilisant un **langage de migration d'instances**. Ce dernier est constitué d'un ensemble de **patterns de migration**. Un pattern de migration capture une **classe de contraintes dépendante de l'application** (*par exemple, compatibilité de l'historique*) et peut être instancié par le fournisseur de service pour définir des règles de migration spécifiques dans le cadre d'une stratégie de migration particulière.

En résumé, la présente thèse apporte les contributions majeures suivantes :

1.3.1 La spécification d'un langage de migration d'instances

Le langage de migration d'instances est basé sur les expressions de chemins et les sur les variables de chemins. Il permet de sélectionner les instances à faire migrer, de définir les contraintes régissant la migration et, enfin de transformer les instances actives vers le nouveau protocole.

Le langage proposé est paramétrable, dans le sens ou il manipule des variables de chemins qui permettent de sélectionner et de migrer des ensembles d'instances spécifiques, suite à un changement d'un protocole de service. Pour raffiner le langage proposé, ce dernier est consolidé par un ensemble de patterns de migration.

1.3.2 La formalisation des patterns de migration

Les structures décrivant les protocoles métier des services web sont analysées afin d'identifier les propriétés pertinentes associées aux opérations de changement des spécifications des protocoles et leurs relations avec les instances actives. Ces propriétés sont utilisées pour formaliser un ensemble de patterns qui expriment les contraintes de migration à respecter par les instances actives. **Les patterns de migration** sont exploitées pour spécifier différentes stratégies de migration.

Ainsi, les patterns de migration formalisés enrichissent le langage de migration proposé, par la prise en compte de différentes dimensions d'analyse, telles que la dimension temporelle (*exécutions passées vs exécutions futures*) et la dimension déontique (*contraintes d'interdiction vs d'obligation*) [26].

1.3.3 Une technique de composition des patterns

Pour permettre aux gestionnaire de protocoles de spécifier des stratégies de migration plus élaborées, une technique de composition des patterns est proposée et formalisée. En effet, comme les stratégies de migration réelles sont non exhaustives et doivent intégrer plusieurs contraintes simultanément, la portée des patterns de base s'avère réduite. Pour répondre à cette préoccupation, la composition des patterns de base répond d'une manière conséquente aux besoins spécifiques et très variés des gestionnaires de protocoles de services. L'élaboration des stratégies de migration est rendue déclarative, dans le sens ou elle est décrite par le choix, le paramétrage et la combinaison des patterns identifiés dans notre approche.

1.3.4 Une étude d'un scénario applicatif

Une étude de cas réelle, inspirée du domaine de la sécurité sociale (processus métier retraite), est explorée le long de la thèse. En réponse aux besoins d'évolution induits par l'environnement, divers exemples d'opérations de changement sont appliqués sur les versions initiales du protocole Retraite. Pour illustrer l'applicabilité de l'approche proposée et afin de répondre aux changements appliqués, différentes traces d'exécution des instances actives sont illustrées et leur migration est analysée conformément aux patterns de migration proposés.

1.3.5 Une implémentation et expérimentation de l'approche

L'approche proposée a été implémentée et expérimentée. Le prototype réalisé permet de gérer différentes phases du cycle de vie d'un protocole de service (*spécification, exécution, évolution, génération des traces,...*), et par la suite, de sélectionner la stratégie de migration adéquate pour effectuer l'analyse de l'impact des changements sur les instances actives. Cet outil logiciel facilite considérablement la gestion de l'évolution des protocoles, en particulier, lorsque le nombre de services invoqués est élevé et lorsque le nombre d'instances actives est très significatif. Les performances globales du prototype réalisé sont expérimentées sur des données synthétiques. Par ailleurs, la complétude et l'expressivité du langage proposé sont évaluées.

1.4 Organisation de la thèse

Le manuscrit est composé de huit chapitres, structurés en deux parties.
Le présent chapitre préambule est une introduction générale qui précise le contexte de l'étude, la problématique de recherche, les contributions de la thèse ainsi que l'organisation du manuscrit.

La première partie de la thèse est un état de l'art du domaine. Elle présente les services web et les processus métiers, ainsi que les travaux connexes ayant abordé la question de l'évolution et de l'analyse de l'impact des changements.

Cette partie est organisée comme suit :

- **Le chapitre 2** rappelle brièvement les définitions, les concepts et les standards de l'architecture et du fonctionnement des services web. Nous y présentons également les apports de cette technologie pour l'intégration des systèmes d'informations hétérogènes et distribués. L'accent sera mis particulièrement sur la notion de protocole des services web et sur leur modélisation.

- **Le chapitre 3** est consacré aux processus métiers et à leurs évolutions. Tout d'abord, nous mettons en exergue l'importance de ce concept dans les organisations modernes, puis nous abordons en détails les systèmes de gestion des processus métiers. Une étude détaillée des changements, de leur raisons et de leurs typologie est exposée. Elle est suivie de l'exposé des différents types d'analyse de l'impact induits par les changements. Nous terminons le chapitre par l'exposé de quelques techniques d'analyse d'impact du changement.

- **Le chapitre 4** englobe un état de l'art détaillé des approches proposées dans la littérature de recherche académique et dans le domaine industriel pour la gestion des évolutions. Une analyse des différents approches et des outils logiciels émanant de diverses communautés est exposée et une étude comparative est illustrée par la mise en évidence des limites des approches existantes pour gérer les différents aspects de l'évolution des protocoles des services web. Les lignes directrices de notre contribution sont dégagées à partir des insuffisances constatées.

La deuxième partie expose notre contribution. Elle est divisée en trois chapitres.

- Dans **le chapitre 5**, nous présentons le fondement formel de notre approche. Nous commençons par détailler notre approche déclarative pour la gestion de l'évolution dynamique des protocoles des services web. Après l'exposé des outils formels nécessaires à la modélisation de l'approche proposée, le langage de migration d'instances est exposé et illustré par des exemples.
- **Le chapitre 6** est dédié à l'identification et à la formalisation des différents patterns de migration et chaque pattern est illustré par un exemple réel. A la fin du chapitre, la technique de composition des patterns permettant l'élaboration des stratégies de migration plus élaborées est abordée. Elle est suivie d'une étude de cas complète qui montre l'applicabilité de notre approche à un scénario réel.
- **Le chapitre 7** décrit la mise en œuvre de notre approche pour la gestion de l'évolution dynamique des protocoles des services web. L'architecture et les fonctionnalités du prototype logiciel implémentant l'approche proposée sont présentées et différents scénarios de gestion de la migration, accompagnés de l'analyse de l'impact de l'évolution, sont illustrés. Par ailleurs, quelques résultats expérimentaux sont donnés et discutés. Nous clôturons le chapitre par une évaluation de la complétude du langage proposé.

En fin, la conclusion générale, donnée au **chapitre 8**, récapitule les contributions de la thèse et esquisse les perspectives ouvertes pour les travaux de recherche futurs.

Première partie

État de l'art

Chapitre 2

Les services web et les protocoles de services

2.1 Introduction

La technologie des services web a émergé lors de la dernière décennie. Elle s'est imposée, aujourd'hui, comme une technologie de choix pour l'intégration des systèmes d'information hétérogènes et distribués. Par ailleurs, en tant qu'implémentation concrète des architectures orientées services (**SOA** [1]) [4, 5], les services web sont la nouvelle génération de composants logiciels, et ils constituent l'aboutissement ultime du processus historique d'évolution des différents paradigmes de programmation [1].

Les concepts et les plate-formes associés aux services web favorisent l'intégration des applications d'entreprises, en offrant un socle technologique commun très adapté à : (i) La réutilisation des applications publiées sur internet ; (ii) la souplesse et l'amélioration des échanges inter-partenaires ; (iii) la construction de services complexes et à plus forte valeur ajoutée, par articulation (*composition*) des éléments simples déjà existants (*services*).

L'avènement de la technologie des services web a suscité beaucoup d'engouement chez les différents acteurs socio-économiques pour le déploiement de leurs applications à grande échelle, en tant que services. En effet, dans les environnements économiques modernes, qui sont de plus en plus ouvert, les processus métiers sont implémentés en tant qu'applications publiées sur Internet comme des services web. Une fois publiées, ces applications peuvent, ainsi, être invoquées et composées par différents partenaires via des protocoles standards afin de répondre à différents besoins de gestion. Par ailleurs, ces applications peuvent dialoguer à distance via Internet, indépendamment des plates-formes et des langages sur lesquelles elles reposent. La conséquence immédiate de ces avancées technologiques est que l'intégration des applications intra et inter-entreprises est considérablement facilitée.

Nous commençons le chapitre par la section 2.2, dans laquelle nous définissons les services web, et nous présentons les concepts de base qui leur sont associés. Ensuite, nous exposons en section 2.3 l'architecture d'interopérabilité les supportant et leur mécanisme de fonctionnement. La section 2.4 traite des protocoles standards manipulés par cette technologie. Nous discutons des techniques de composition des services en section 2.5, et après nous exposons les protocoles des services web et leurs modèles de représentation en section 2.6. Nous passons en revues quelques questions résiduelles soulevées par la communauté de recherche dans le domaine des services web en section 2.7, et nous clôturons le chapitre par une conclusion en section 2.8 qui synthétise les apports de cette technologie.

1. **S**ervice **O**riented **A**rchitecture

2.2 Définition et concepts de base

Dans les environnements service web, chaque service représente une fonctionnalité réutilisable par des humains, par d'autres applications ou par d'autres services, sans avoir à connaître ni les détails de son implémentation ni la façon dont le service a été construit. Puisque les protocoles du web sont complètement indépendants des fournisseurs, des plateformes et des langages de mise en œuvre, les applications supportées par les technologies de services web sont à la fois intégrables, et suffisamment flexibles pour faire face aux modifications potentielles qui affectent les environnements des entreprises. En plus, chaque service est considéré comme une brique de base pouvant être intégrée dans le cadre d'une composition de services, et ce afin de créer de nouvelles applications à plus forte valeur ajoutée [27, 28, 29].

2.2.1 Définition des services web

De façon très simpliste, un service web peut être défini comme : "*un composant logiciel qui réalise une fonction applicative qui est accessible par d'autres applications (un client, un serveur ou un autre service web) à travers le réseau Internet. Ce service applicatif peut être implémenté comme une application autonome ou comme un ensemble d'applications*" [1].

Cependant, la définition précédente est très ouverte car elle considère un service web, simplement comme, une application accessible à d'autres applications à travers le web. En ce sens, elle sous entend que tout objet possédant un **URL** (**CGI**, **API**) est un service web [1]. Néanmoins, la différence fondamentale est que les services web s'appuient sur un ensemble de protocoles qui standardisent les modes d'invocation mutuels des composants applicatifs [30].

Dans la littérature différentes définitions ont été proposées pour spécifier la notion de service web, et dont certaines sont plus généralistes que d'autres. Une définition plus précise est fournie par le W3C. [31]

Définition 2.1 *Un service web est un système logiciel destiné à supporter l'interaction ordinateur-ordinateur sur le réseau. Il a une interface décrite en un format traitable par l'ordinateur (particulièrement WSDL[6]). Les autres systèmes réagissent réciproquement avec le service web d'une façon prescrite par sa description en utilisant des messages SOAP [7], typiquement transmis avec le protocole HTTP et une sérialisation XML, en conjonction avec d'autres standards relatifs au web.*

Cette définition met en exergue l'aspect d'interopérabilité et le besoin de standardisation des protocoles assurant les échanges. En plus, elle permet de dégager au moins, trois principes fondateurs de la technologie des services web qui sont :

a) Les services web interagissent à travers l'échange des messages encodés en XML ; un standard très répandu et largement adopté.

b) Les interactions dans lesquelles les services web peuvent s'engager sont décrites au sein d'interfaces.

c) Des protocoles universels constituent la superstructure permettant la publication, la découverte et l'invocation des services web.

Pour concrétiser les objectifs cités ci-dessus, l'implémentation actuelle des services web repose sur un ensemble de protocoles standards proposés par le W3C (*cf., section. 2.4*).

En définitif, un service web désigne toute application (*programme*) mise à disposition sur Internet par un fournisseur de service, et qui est accessible par les clients via son interface à travers des protocoles Internet standard. Les fonctionnalités du service peuvent être découvertes, invoquées et composées pour fournir une solution ou une réponse à un problème ou à une requête d'un utilisateur qui y accède via l'ubiquité des protocoles web.

Il est important de constater, dés maintenant, que la description de l'interface de service avec le standard WSDL ne permet d'exprimer que des noms d'opérations et des types de messages. Par l'utilisation d'une telle spécification décrite par le standard WSDL, il s'ensuit que la définition précédente restreint la portée des interfaces des services web aux seuls aspects fonctionnels et structurels. Cette insuffisance, ainsi que les solutions apportées pour y remédier seront discutées dans la section relative aux protocoles de services 2.6.

2.2.2 Concepts associés aux services web

Les principaux concepts manipulés dans une architecture services web sont les suivants [1] :

a) **le fournisseur du service** : désigne le serveur qui héberge les services déployés ;

b) **le client du service** : représente l'application cliente qui invoque le service ;

c) **le service** : désigne les fonctionnalités d'un agent logiciel qui implémente le service ;

d) **la description du service** : c'est la spécification du service exprimée dans un langage de description interprétable par les machines ; c'est à dire une description technique dans laquelle le service est vu en termes de messages, de types, de protocoles de communication et d'une adresse physique ;

e) **les messages** : c'est la plus petite unité d'échange entre les clients et les services. La structure des messages qui permettent l'invocation d'un service doit être exprimée dans la description du service. Un atout du modèle de messages est qu'il permet d'abstraire l'architecture du langage, ou encore de la plate-forme qui va supporter le service. Il suffit juste que le message respecte une structure donnée pour qu'il puisse être utilisé.

f) **la ressource** : désigne l'identifiant du service, c'est-à-dire son URI. Bien que le web soit constitué de plates-formes totalement hétérogènes, où les intérêt des différents acteurs du marché s'entremêlent, ceci ne l'a pas empêché de se développer et d'être universel. Ce succès est dû essentiellement à l'édition d'un ensemble de standards ouverts, dont les plus connus sont le protocole HTTP et le format MIME. Ensembles, ils offrent un mécanisme d'échange de données de toutes sortes quelle que soit la nature des plates-formes impliquées.
Le développement des services web a suivi la même approche en mettant en place une campagne de standardisation qui a touché les aspects les plus importants du développement d'un modèle d'intégration d'applications hétérogènes.

L'avantage d'une architecture logicielle à base de services web est de présenter les services comme des boites noires. En fait, les entrées/sorties d'un service sont gérées au sein de messages dont nous connaissons le format grâce à des interfaces clairement exposées, mais sur lesquelles l'implémentation interne du traitement n'influe pas au niveau de la structure. Ceci permet d'atteindre un haut niveau de modularité et d'interopérabilité favorisant la réutilisation des services ainsi que leurs composition.

2.3 Architecture et fonctionnement des services web

L'intégration des applications dans les environnements services web pose des problèmes complètement différents de ceux rencontrés dans le cadre des middlewares conventionnels (*RPC, CORBA, DCOM, JRMI,...*). En effet, dans le contexte des services web, le problème ne se limite plus à celui de la communication inter-systèmes hétérogènes, mais de nouvelles contraintes conceptuelles et des implications architecturales résultent de l'utilisation d'Internet comme canal de communication qui connecte les systèmes à intégrer. Il en résulte que l'exploitation effective des potentialités offertes par la technologie des services web passe, inévitablement, par l'adoption d'un ensemble de normes universelles pour les échanges inter-partenaires.

La description de l'architecture et du fonctionnement des services web consiste à mettre en relation les concepts précédents, à décrire la dynamique d'interaction existante entre les divers éléments manipulés, ainsi que la description de l'ensemble des contraintes qui les gouvernent.

2.3.1 Architecture des services web

L'originalité de l'infrastructure des services web se manifeste par la mise en place des services sur la base exclusive des protocoles les plus répandus sur Internet. Ces protocoles sont répartis en plusieurs couches, telles que représentées dans la figure 2.1, proposée par le W3C [31].

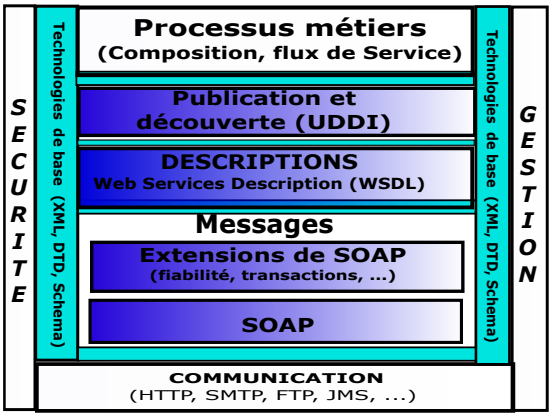

FIGURE 2.1 – Pile conceptuelle des services web selon la norme W3C

- **Couche de transport (communication)** : cette couche s'occupe de transporter les messages entre applications en utilisant les protocoles HTTP, FTP ou SMTP ;
- **Couche des messages en XML** : elle permet de formaliser les messages à l'aide d'un vocabulaire XML commun (**SOAP**) ;
- **Couche de description des services** : c'est la description de l'interface publique des services web (**WSDL**) ;
- **Couche de recherche des services** : s'occupe de la centralisation des services et de leur description dans un référentiel commun (**UDDI**).

Cette architecture de base a fait l'objet de plusieurs enrichissements émanant de plusieurs communautés. En effet, au dessus de la couche publication et découverte, deux autres couches ont été empilées. La première couche supplémentaire permet de décrire les processus métiers associés aux services web, par la prise en charge de modèle de description spécifiant la logique métier manipulée, alors que la deuxième couche se focalise sur les services web sémantiques [32, 33].

En se basant sur les protocoles standards, les acteurs impliqués dans les environnements services web interagissent par l'échange de messages. Les rôles de ces acteurs sont les suivants :

1. **Le fournisseur de service** : C'est le propriétaire du service. Il publie le service au niveau de l'annuaire de service, en déposant le fichier de description dans un format préalablement défini et contenant les informations commerciales, et aussi les services offerts ainsi que les informations nécessaires à l'invocation du service. D'un point de vue technique, il est constitué par la plate-forme d'accueil du service.
2. **Le client ou demandeur** : C'est le consommateur de service. Il recherche le service désiré dans l'annuaire, sélectionne un service et l'invoque à travers la description publiée en interagissant directement avec le fournisseur. D'un point de vue technique, il est constitué par une application (*ou un utilisateur*) ou même un autre service web.
3. **L'annuaire du service** : Correspond à un registre de descriptions de services offrant des facilités de publication de services à l'intention des fournisseurs et des facilités de recherche de services à l'intention des clients.

La figure 2.2, décrit les relations entre ces trois acteurs.

Les trois opérations de base de cette architecture sont :(i) la publication de la description ; (ii) la localisation ou la découverte du service, suite au déclenchement de la recherche, et enfin (iii) l'interaction entre le client et le fournisseur (*binding*).

La description détaillée de ces opérations est donnée dans ce qui suit :

1. **La publication** : Les services web sont centrés autour du concept de service. La première opération, consiste à le décrire. Les messages associées aux opérations offertes par le service ainsi que leur signature (*entrées/sortie*) sont spécifiés en détails. Vu que les services sont dénués de contexte implicite à cause de l'absence d'un médiateur central (*contrairement aux middlewares conventionnels*), il est impératif de spécifier son adresse (**URI**) et son protocole (**e.g. HTTP**). Une fois ces informations formalisées dans le langage WSDL, le fichier de description doit être publié dans l'annuaire de services pour d'éventuelles recherches.

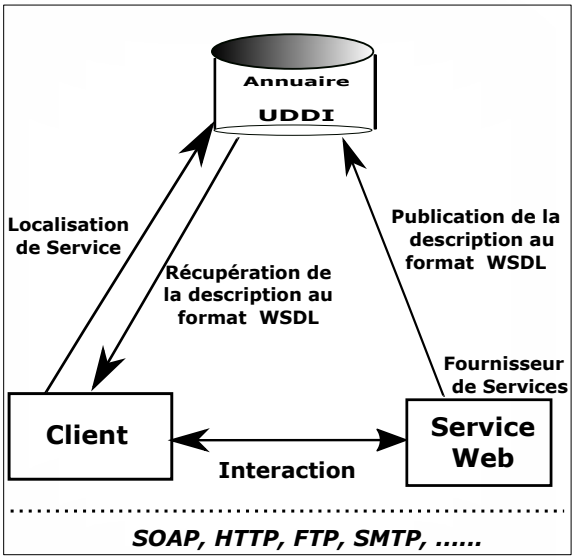

FIGURE 2.2 – Les acteurs et les opérations des services web

2. **La découverte du service** : L'annuaire de services stocke les descriptions des services (*fichiers WSDL*) qui étaient antérieurement enregistrés. Ainsi, l'exploitation de cet annuaire permet aux utilisateurs de rechercher les services répondant à leurs besoins, et de les localiser sur la base d'un ensemble de critères de recherche.
3. **L'interaction *(binding)*** : Une fois le service est localisé, l'opération de binding consiste en son invocation par le client, et en la réponse du fournisseur à cette requête dans une session de conversation entre les deux acteurs. C'est le mécanisme d'interaction avec le service.

2.3.2 Fonctionnement des services web

Le fonctionnement des services web est matérialisé par les cinq interactions dans lesquelles les trois acteurs précédents peuvent s'engager. Ces interactions sont décrites dans la figure 2.3.

1. **La publication du service** : Après avoir décrit le service dans le format WSDL, le fournisseur doit le publier dans l'annuaire de services (**UDDI**) afin de permettre aux futurs utilisateurs de prendre connaissance de sa disponibilité. Cette opération consiste en la fourniture des informations concernant le fournisseur, telles que les informations commerciales, et d'autres qui sont relatives au service lui-même (*les fonctions assurées*) ainsi que les informations techniques (*méthode d'invocation, adresse, protocole ...*).
2. **La recherche du service** : Le client désirant utiliser la technologie des services web doit interroger l'annuaire (**UDDI**) par l'envoi d'une requête encapsulée dans une enveloppe SOAP, afin de rechercher le(s) service (s) répondant à ses besoins. Différentes approches ont été proposées dans la littérature pour assurer

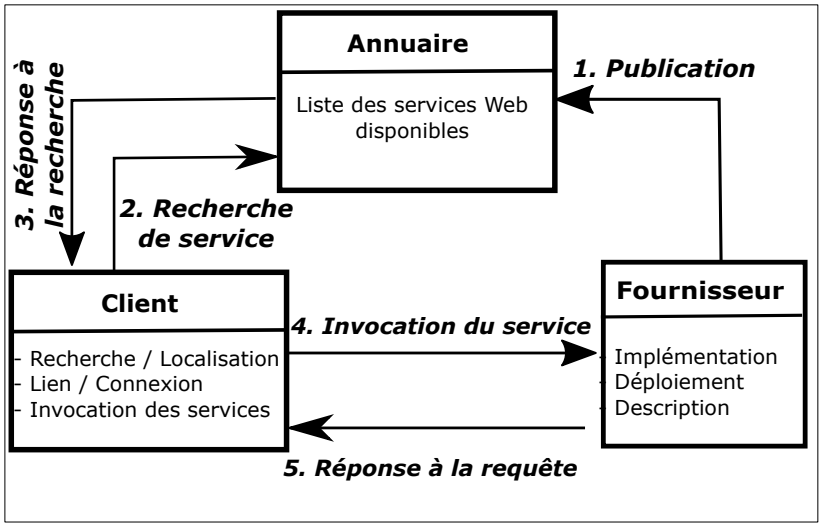

FIGURE 2.3 – Fonctionnement des services web (*d'après F. Casati et al.* [1])

la découverte des services web (*voir [34, 35], par exemple)*. Dans le cas où plusieurs résultats sont fournis en réponse à la requête de recherche, des techniques de sélection de services sont offertes et peuvent être adoptées (*premier trouvé, aléatoirement, sémantique, ...*) [36, 37, 38] ou bien sur la base de paramètres techniques, tels que la qualité de service (QoS) [39, 40].

3. **Réponse à la Recherche** : Le résultat de la requête de recherche dans l'annuaire de service est un fichier WSDL qui sera transmis dans un message SOAP au client demandeur. Ce fichier contient la description du service et les informations techniques nécessaires à son invocation.
4. **Invocation du service** : Une fois ayant le fichier WSDL en sa possession, le client interagit directement avec le fournisseur du service. Il envoi un message SOAP, contenant, en plus du (des) nom(s) de(s) l'opération(s) à exécuter et de ses paramètres, l'adresse (**URI**) et le type de protocole de communication (**HTTP**) utilisé. Techniquement, cette invocation est basée sur à un appel de procédure à distance (**RPC**).
5. **Réponse à l'invocation** : Le fournisseur réagit à la réception du message émanant du client par l'exécution du programme (*stub-serveur*)concerné, et il envoie au client la réponse résultante de l'exécution de la procédure invoquée. Cette réponse est transmise, à son tour, sous la forme d'un message SOAP.

2.4 Les protocoles standards des services web

Le premier effort permettant un déploiement réel des services web a consisté en la standardisation des langages et des protocoles utilisés. Comme il a été réalisé pour le web lui-même (*HTTP, HTML, navigateur, serveur web...*), la nécessité de continuer cet effort permettra non seulement le fonctionnement, mais aussi la prolifération des services web. Ainsi, la spécification des normes faisant fonctionner les services doit

demeurer aussi génériques que possible, afin de permettre leur exploitation par divers partenaires. A cet effet, des standards universels à chaque niveau de la pile conceptuelle de la figure 2.1 sont inévitables.

Dans cette section, nous allons exposer les standards associés aux services web, en commençant par le langage XML, comme socle de base commun. Pour chaque standard, on donnera sa définition et ses objectifs. Pour plus de détails sur les normes et les langages concernés, le lecteur pourra se référer aux spécifications techniques de chaque standard : [6] pour WSDL, [7] pour SOAP et [8] pour l'UDDI, ou il pourra consulter des références adéquates dans le domaine, telles que [1].

2.4.1 XML : Extensible Marckup Language

XML [41] est un langage de balisage à mi-chemin entre la complexité exagérée de SGML [42, 43, 44] et le balisage figé de HTML [45, 46]. C'est un format de représentation de données et d'échange de documents sur Internet qui est basé sur la syntaxe au lieu de la sémantique. Sa simplicité, sa portabilité sur différentes plateformes, son auto-description ainsi que son extensibilité ont conduit à sa grande popularité et son adoption comme standard d'échange à travers le web. Plus particulièrement, comme langage de base commun autour duquel s'articulent les services web.

Exemple de document XML représentant les données d'un compte bancaire :

```
< ?xml version="1.0" encoding= "UTF-8"? >
    <Compte>
        <montant> 1000 </montant>
        <monnaie> dinars </années>
        <années> 20 </années>
        <taux-intérêt> 4.5 </taux-intérêt>
        <Taxe-annuel> 200 < Taxe-annuel>
    </Compte>
```

Un document XML est un arbre composé d'un ensemble d'éléments structurés en balises. Cette structuration hiérarchisée ouvre la voie au traitement automatique du document. Par ailleurs, avoir une structure clairement définie et extensible favorise la prolifération d'outils de traitements des documents XML pour l'extraction de leur structure, aussi bien que de leur contenu. Aussi, le test de validité d'un document par rapport à une structure préalablement définie (*DTD ou XML schema*) est alors possible. Il existe une panoplie d'outils d'exploitation des documents XML (*Xpath, Xquery pour interroger des documents, XSL, XSLT pour transformer des documents,...*).

Un ensemble de standards basé sur XML et traitant des différents aspects inhérents au déploiement des services web s'est développé et se développe encore. Ces spécifications émanant de constructeurs et consortium divers permettent de construire des services web, d'assurer l'interopérabilité entre les composants applicatifs et de garantir des conditions opérationnelles d'exploitation réelle, de façon à :

- Effectuer les échange de données inter-entreprises : xCBL, RosettaNet, ..., etc.
- Assurer le fonctionnement des services web : SOAP, WSDL, UDDI...
- Assurer l'intégration des processus collaboratifs sur le web (*Workflows*) : BPML, XLANG, BPEL, ..., etc.
- Fournir des mécanismes de sécurité : XKMS, ..., etc.

2.4.2 SOAP : Simple Object Access Protocol

C'est un protocole proposé par le W3C, pour assurer interopérabilité entre composants, tout en restant indépendant des plateformes et langages. Il assure des appels de procédures distantes (**RPC**) en utilisant le langage XML et le protocole HTTP (*d'autres protocoles comme SMTP peuvent être utilisés, mais HTTP est le plus populaire*).

Un message SOAP est un document XML encapsulé dans une enveloppe permettant d'organiser les informations de manière à ce qu'elles puissent être échangées entre partenaires. Le protocole SOAP a pour objectifs : [1]

- Offrir un format de messages pour les communications entre partenaires, par la description des informations à échanger dans une structure XML.
- Définir les conventions nécessaires à l'appel de procédures à distance pour invoquer un service par un client, en envoyant un message SOAP, et comment le service répond par un autre message SOAP.
- Décrire comment un message SOAP doit être transporté au dessus de HTTP ou SMTP.

Il s'agit, donc, d'un mécanisme fondamental pour assurer l'interaction et le dialogue entre applications distribuées dans un environnement services web. En exploitant les mécanismes de ce standard, clients et fournisseurs peuvent formuler, transmettre et comprendre les messages SOAP échangés.

2.4.3 WSDL : Web Services Description Language

Le standard WSDL décrit les interfaces des services web dans le format XML. Il est similaire à l'ancien IDL des middlewares conventionnels. En plus de la spécification des opérations offertes par le service, WSDL explicite aussi les mécanismes d'accès au services web (*protocole de communication*) et sa localisation (**URI**), afin de préciser où envoyer les messages SOAP. En offrant une nette séparation entre l'interface de service et les mécanismes d'accès (*binding*), différents services peuvent implémenter la même interface, mais seront disponibles à des adresses différentes et peuvent être invoqués avec des protocoles différents [1].

Il est clair que les spécifications WSDL facilitent le déploiement des services web en les rendant "*auto descriptifs*". En effet, elles leur permettent de décrire de manière abstraite et indépendante du langage de programmation ce qu'ils font, comment ils le font, et comment les clients peuvent exploiter les fonctionnalités offertes.

En résumé, les objectifs de WSDL sont : [1]

- Décrire les interfaces des services en précisant les opérations offertes et leur signatures (*paramètres d'entrées/sorties et types*). Ces interfaces constituent les contrats que les clients doivent respecter pour pouvoir interagir avec le service.
- Préciser le (s) protocole (s) d'accès au service (**HTTP, SMTP. . .**).
- Fournir la localisation du service où il peut être invoqué (**URI**).
- Définir une description de la sémantique du service par la fourniture des informations permettant aux développeurs de traiter le contenu des services.

Cependant, avec la spécification actuelle de WSDL (*version 1.2*), ce dernier objectif reste loin d'être atteint. Pour surmonter cette limitation, les développeurs font recours

à d'autres mécanismes en dehors de WSDL pour rattacher de la sémantique à leurs services (*voir sections 2.6 et 2.7.2*).

2.4.4 UDDI : Universal Description Discovery and Integration

Contrairement aux standards précédents qui ont été proposés par le W3C, le standard UDDI est né de la collaboration entre plusieurs entreprises dont IBM, Microsoft et Ariba. Actuellement, il est géré par OASIS [2]. En assurant le rôle d'un médiateur centralisé pour l'ensemble des partenaires désirant faire des échanges à l'échelle du web, l'annuaire de services répond parfaitement aux besoins des fournisseurs des services pour la publication de leurs services et aux exigences des clients recherchant des services sur le web.

Les objectifs de l'annuaire UDDI sont : [1]

- Offrir un cadre pour la description et la découverte des services web, en fournissant des structures de données et des APIs, pour la publication des descriptions des services dans les registres adéquats et pour la découverte de ces publications par les clients.
- Guider et soutenir les développeurs dans la recherche des informations concernant le service, afin qu'ils puissent réaliser des programmes clients pouvant interagir correctement avec ces services.
- Permettre les liens (*binding*) dynamiques, en offrant aux clients la possibilité d'interroger le registre UDDI et de récupérer les références des services (*description WSDL*) qui les intéressent.

2.5 Composition des services web

Le problème de la composition des services se pose quand une requête d'un client ne peut être satisfaite par aucun des services disponibles [47]. Il devient, donc, nécessaire de combiner les services existants afin de créer un nouveau service à plus forte valeur ajoutée et répondant au besoin spécifique du client demandeur. Ainsi, l'implémentation du processus métier d'un service web composite implique plusieurs services composants élémentaires [48]. Par ailleurs, une composition de service est un processus récursif du fait qu'il peut impliquer, soit des services élémentaires soit des services entiers, qui sont à leur tour des services composés [49].

2.5.1 Définition de la composition des services

La composition des services web services vise à engager une collaboration entre différents services pour réaliser une tache qui ne peut être accomplie par l'exploitation séparée de chaque service [49].

Dans cette perspective, la composition vise à simplifier, de façon considérable, le processus de développement rapide des applications en favorisant la réutilisation des

2. OASIS est une organisation indépendante regroupant plus de 300 compagnies du monde informatique

composants existants [27]. En effet, un service composé est obtenu en combinant "des parties" des services composants disponibles [47].

A titre d'exemple, si un utilisateur désire organiser un voyage, il n'est pas suffisant pour lui d'acheter un billet d'avion, mais il doit aussi réserver un hôtel, louer une voiture, se divertir .... Dans ce scénario, l'utilisateur pourra réaliser sa tâche en profitant des possibilités offertes par la panoplie des services web publiés. Dans cette perspective, il opère une composition des divers services existants. Une telle composition peut être gérée manuellement, ce qui signifie que c'est l'utilisateur lui même qui sélectionne les services qu'il désire exécuter un par un, comme elle peut être lancée d'une manière automatique.

2.5.2 Les buts de la composition

Les buts de la composition des services dépendent des objectifs attendus, et qui peuvent être les suivants : **la synthèse, la vérification** ou **la supervision**.

La synthèse des services [50] consiste à générer les spécifications des services composites en combinant ceux existants. La vérification de la composition vise à vérifier si un service web satisfait certaines propriétés spécifiques (*qu'il assure la sécurité, par exemple*) [51, 52]. La supervision [53] est déclenchée lors de l'exécution des services composites, et elle permet d'analyser différents paramètres d'exécution (*par exemple : le temps, le cout*).

2.5.3 Processus de composition : manuel vs automatique

La composition manuelle est basée sur l'intervention d'un expert humain. Elle aborde et traite les questions de programmation et d'exécution de bas niveau. Un exemple des environnements de composition manuelle inclus BPEL [54] et Microsoft BizTalk[3]. La composition manuelle est souvent complexe, pose des problèmes de passage à l'échelle et elle est potentiellement porteuse d'erreurs.

La composition automatique simplifie considérablement le processus de composition, et elle offre des possibilités pour personnaliser les systèmes complexes qui sont construits à la volée en fonction des besoins formulés par l'utilisateur. Elle aboutit à des applications flexibles avec une grande réactivité aux défaillances et à l'adaptation dynamique aux changements de contexte. Par ailleurs, des techniques ont été développés pour permettre la vérification de la composition des services web générés, que ce soit automatiquement ou manuellement. Cependant, la composition automatique a été et reste encore, une tâche difficile à réaliser. Nous pouvons citer diverses sources de complexité, telles que : (i) la difficulté (*par exemple : la décidabilité [55]*) de la composition qui dépend étroitement de l'expressivité des modèles de services utilisés et de l'objectif de la composition ; (ii) le grand nombre de services web disponibles sur le web ; (iii) et la diversité des modèles de représentation des services à cause des besoins de modélisation et/ou à cause de la vision des développeurs.

3. http ://www.microsoft.com/biztalk/en/us/overview.aspx

2.5.4 Nature de la composition : statique vs dynamique

La composition statique se déroule au moment de la conception d'une application du fait que les composants impliqués sont choisis, liés et assemblés avant d'être déployés [9]. Une telle approche de composition est adaptée pour les environnements fermés et caractérisés par une évolution peu fréquente des composants.

Par contre, la composition dynamique s'effectue au moment de l'exécution (*at run-time*), et elle permet de créer de manière autonome des services complexes, en combinant à la volée les composants en fonction des demandes et du contexte de l'utilisateur [56]. Ce dernier type de composition est opéré dans les environnements ouverts et flexibles, où la sélection et la combinaison des composants sont faits à la demande. L'approche de composition dynamique des services est, généralement, assujettie à divers défis, tels que : (i) le grand nombre de services quotidiennement disponibles ; (ii) le nombre sans cesse croissant de fournisseurs de services ; (iii) la nature volatile des services web (*par exemple, ils peuvent disparaître, peuvent être modifiés ou temporairement indisponibles*).

2.6 Les protocoles métiers des services web

Comme il a été expliqué dans la section 2.4.3, les interfaces statiques des services web, telles que décrites par les fichiers WSDL, sont indispensables pour faciliter l'utilisation des technologies à base des services. Effectivement, elles offrent un mécanisme essentiel pour l'interopérabilité entre les divers intervenants. Néanmoins, ces interfaces restent insuffisantes, et il y a un besoin réel d'abstractions de haut niveau au delà des simples "format de données et noms des messages". En effet, dans un environnement réel les clients n'invoquent pas de simples opérations qui sont indépendantes de leur contexte, plutôt ils exécutent un ensemble d'opérations dans un ordre bien défini afin de réaliser des processus métiers. Il devient alors impératif de prendre en considération les aspects comportementaux des services web. A ce titre, il existe un consensus sur la nécessité de préciser également l'interface dynamique des services [10, 57, 58, 59]. En particulier, le comportement extérieur et visible d'un service est important pour éviter des conversations incorrectes entre le service et ses clients [1, 10].

A titre d'exemple, un service peut offrir les deux opérations suivantes : ajouter des produits à un panier électronique et procéder au paiement. Un document WSDL d'un tel service doit expliciter les noms des messages associés (*par exemple, ajouter-produits et paiement*), ainsi que leur schéma XML (*type, port types, noms d'opérations, messages, ...*). Il est évident qu'une description aussi simpliste demeure insuffisante pour réaliser de vrais processus métiers. Par exemple, la suite d'opérations : payement suivie de de l'opération ajouter-produits n'exprime pas souvent un processus métier réel. Par conséquent, les opérations offertes par un service web ne peuvent être exécutées dans un ordre aléatoire, et il est rare que ces opérations soient invoquées indépendamment les unes des autres. En effet, les interactions entre les clients et les services sont structurées en termes d'un ensemble d'opérations qui doivent obéir à des contraintes à respecter par les clients afin qu'ils puissent réaliser les procédures désirées [10].

Dans cette section, nous rappelons le concept de protocole de service et son utilité, après nous discutons des différents modèles existants pour leur représentation.

2.6.1 Notion de protocole de service

Un protocole de service est un modèle abstrait très utile pour capturer le comportement externe et visible d'un service web. Il explicite un ensemble de contraintes à satisfaire par les clients qui invoquent le service [10, 11]. En ce sens, il spécifie les conversations qu'un service peut supporter afin que les demandeurs potentiels sachent comment éviter d'invoquer des conversations invalides [60]. Dans [10], des contraintes sur l'**ordre des messages** que le service web peut accepter ont été spécifiées. Les travaux présentés dans [61] se focalisent sur les **contraintes temporelles** (*durée de validité*) imposées à chaque activité du protocole. Un enrichissement des deux types de contraintes précédentes a été proposé dans [62] pour permettre la prise en compte des **contraintes transactionnelles**.

Exemple 2.1 *La figure 2.4 montre un exemple simple d'un protocole d'un service web d'*achat en ligne *qui pourra être déployé par un fournisseur de service pour gérer les commandes clients relatives au commerce électronique. Le protocole est représenté sous forme d'un automate d'états fini déterministe, où les états correspondent aux différentes étapes par lesquelles passe le processus d'achat électronique (*Connecté, CommandeConfirmée, CommandePayée ; . . . *) et les transitions entre états correspondent aux noms des opérations invoquées (*Commander, Payer, Annuler . . . *).*

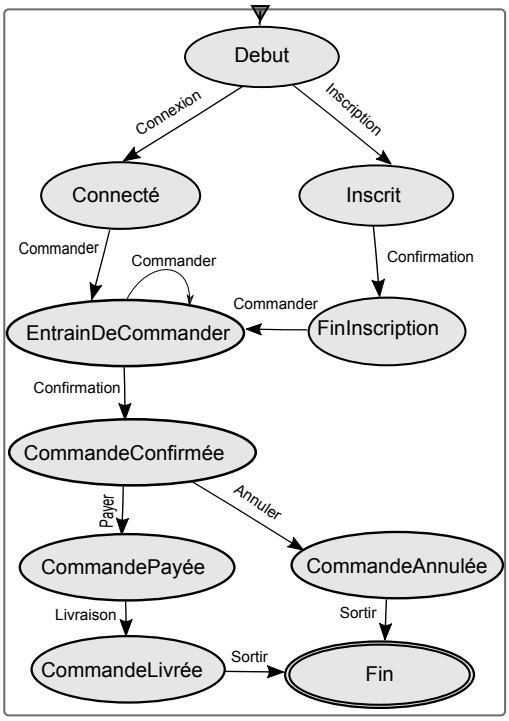

FIGURE 2.4 – Potocole du service web Achat en ligne

La logique métier véhiculée par ce protocole stipule que le client doit tout d'abord se (Connecter), ou s'(Inscrire) s'il ne l'est pas déjà. Puis, il pourra (Commander) des

articles, tout en restant dans l'état (EntrainDeCommander). Après confirmation (Confirmation), il a la possibilité, soit d'(Annuler)) sa commande soit de la (Payer). Si la commande a été confirmée alors la marchandise sera livrée (*opération :*Livraison). Enfin, le service transite à l'état final, suite à l'exécution de l'opération (Sortir) pour atteindre l'état final (Fin).

Il est a noter que dans le modèle présenté ci-dessus, les noms des états n'ont pas le même degré d'importance et de signification que les transitions. En fait, ces dernières sont plus importante, car elle expriment les messages à échanger entre les clients et les fournisseurs.

2.6.2 Utilité des protocoles de service

Les protocoles de service web sont des outils très utiles dans plusieurs contextes d'analyse relatifs aux phases de développement et d'exécution des services [11, 60]. Ils sont, particulièrement, incontournables dans les situations suivantes :

- Vérification de la conformité des protocoles des fournisseurs : s'assurer que la mise en œuvre effective d'un service répond à un cahier des charges qui est fourni aux clients (*par exemple, dans la documentation*).
- Vérification de la compatibilité des protocoles fournisseurs avec des normes standards, telles que RosettaNet PIPs [63].
- Pour le client, vérifier que son propre protocole est en conformité avec la logique métier du fournisseur. Dans le cas contraire, le client sera obligé d'effectuer les adaptations nécessaires de façon à rendre son protocole **compatible** avec celui du fournisseur, et éviter, par conséquent, les erreurs d'exécution.
- Il existe plusieurs situations dans lesquelles le service web aura besoin d'être remplacé (*cas de panne du service, lors des évolutions des procédures, ...*). Dans de telles scénarios, l'analyse de la **remplaçabilité** du service, basée sur les protocoles de service [11, 64], permettra d'identifier de nouveaux services potentiels pouvant remplacer le service défaillant.

2.6.3 Les modèles de représentation des protocoles de services web

Vue l'importance du concept de protocole métier une panoplie de modèles dotés de niveaux d'expressivité variés, a été proposée dans la littérature pour les représenter. Certains modèles sont formels, tels que les automates ou les réseaux de Petri [52] [65]. D'autres modèles sont plutôt graphiques, comme les diagrammes de séquences ou les diagrammes d'activités d'UML [66]. Par ailleurs, même certains langages intègrent dans leurs spécifications des primitives et des outils pour décrire le comportement des protocoles de services ou des processus métiers. C'est le cas par exemple de BPEL [54] et de BPMN [67].

A titre d'illustration, nous exposons dans les figures 2.5 et 2.6 deux protocoles de services utilisant des modèles de représentations différents. Le premier exemple de la figure 2.5 est un modèle basé sur les réseaux de Petri qui représente le protocole de service *Réservation en ligne* d'un vol.

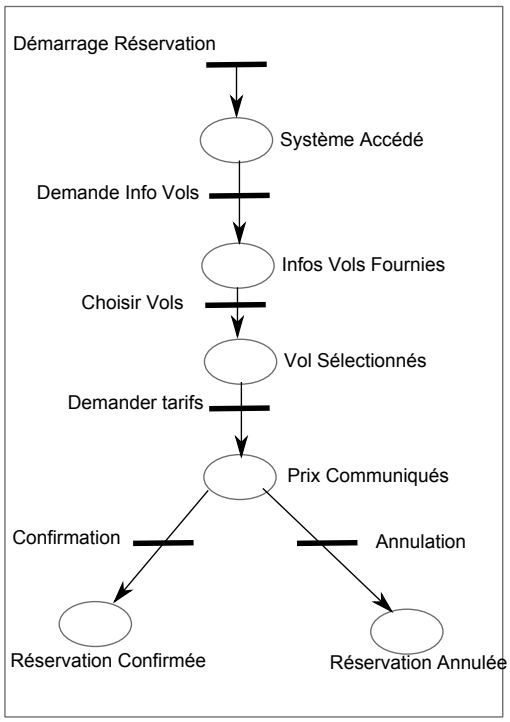

FIGURE 2.5 – Potocole de Réservation en ligne en réseaux de Petri

Le deuxième exemple de la figure 2.6 schématise un diagramme de séquences UML [68] qui illustre le protocole de réservation d'un vol par l'intermédiaire d'une agence de voyage (*fournisseur*). Ce modèle met en évidence les rôles (*colonnes*) et les messages (*flèches étiquetées*) comme éléments conceptuels de base. Quant aux contraintes d'ordre, elles sont exprimées par la chronologie temporelle descendante.

D'une manière générale, nous pouvons classer les modèles de représentation des protocoles des services web en trois grandes familles.

1. **les modèles à base d'états :** sont rencontrés dans plusieurs contexte pour décrire les abstractions comportementales d'un système. Particulièrement, les modèles UML [68] contiennent différentes abstractions qui permettent de décrire différentes aspects d'un système en fonction du type d'information qu'on désire capturé. Ainsi, les diagrammes de séquence, les diagrammes d'activités et les diagrammes d'états représentent différents comportements d'un système.
2. **les modèles à base d'activités :** sont utilisées pour représenter un système dans sa forme exécutable (*e.g, workflow*). Les états représentent les activités et les transitions sont déclenchées quant les activités sont achevées. Dans certains de ces modèles, les transitions véhiculent des flux de données.
3. **les modèles à base de règles :**[69] spécifient le comportement d'un système par le biais d'un ensemble de règles. Une règle est déclenchée lorsqu'un événement donné se produit, ou lorsqu'une condition est vérifiée. La règle est écrite dans un langage spécifique et elle définit une action à exécuter. Les modèles à base de règles ont été largement utilisés lorsque le comportement du système exige des mises à jour fréquentes à partir de l'expertise du domaine.

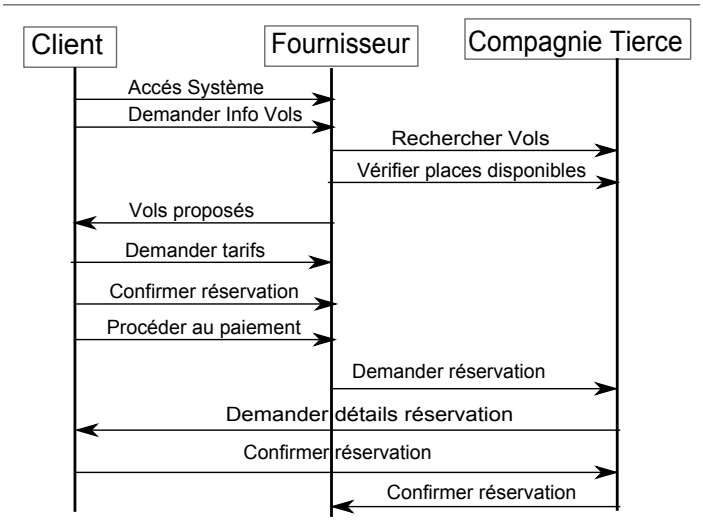

FIGURE 2.6 – Diagramme de séquences du potocole de Réservation en ligne

2.7 Difficultés et défis liés aux services web

Aujourd'hui, les services web se sont imposés en tant que standard *de facto* dans le domaine du génie-logiciel. Mais malgré les avancés spectaculaires réalisées par cette technologie, certaines difficultés et défis demeurent résiduels. Cette section comporte une discussion qui passe en revue certaines questions inhérentes à cette technologie, et qui constituent actuellement des thématiques de recherche très actives.

2.7.1 La prolifération des standards

Le principe fondateur de la technologie des services web est la standardisation qui est la pierre angulaire de toute l'architecture SOA. Cette standardisation est exigée à différents niveaux de la couche protocolaire de la figure 2.1. Pour garantir le contrôle de ces normes, les organismes internationaux et consortiums (W3C, OMG, OASIS,...) ont pour mission de valider et de contrôler l'évolution des différentes normes proposées. Néanmoins, la prolifération grandissante de ces standards risque de compromettre le principe lui même, et d'engendrer, par là même, l'effet inverse de ce qui est visé par la normalisation. L'exemple le plus frappant est celui des normes de sécurité pour lesquelles, au moins, une dizaine de standards est proposée ; désignées par l'acronyme **WS-***, tels que : WS-Security [70], WS-Privacy [71], WS-Trust [72], WS-Secureconversation [73], WS-Policy[74], WS-Federation [75], ... Cependant, ces protocoles n'arrivent pas à s'imposer vue l'absence de consensus de la part des différents acteurs.

Le constat précédent est aussi valable dans le contexte de la composition des services web, où il est observé qu'une panoplie de langages de composition est proposée, par exemple : WSFL [76], WSCI [77], WSCL [78], BPEL [54], ...

2.7.2 Les services web sémantiques

La tendance des services web sémantique [32] a stimulé beaucoup d'engouement chez les chercheurs et les industriels pour la découverte, la sélection et l'invocation de services publiés, sur la base de critères sémantiques. Certains travaux envisagent même la composition automatique basée sur la sémantique [79, 80, 81]. L'objectif visé par les environnements des services web sémantiques est de permettre la prise en charge des besoins spécifiques des utilisateurs par des ordinateurs au lieu des humains. Ils projettent d'automatiser d'une manière transparente, une grande variété des processus métiers sur le web, le plus rapidement et le plus efficacement possible [33]. Toutefois, les services fondés sur la sémantique ne sont guère adoptés ni utilisés concrètement. Comme il y a peu d'exemples réels qui démontrent les possibilités et les avantages de ces services. En outre, il y a un manque de cadres de création de services et de plates-formes techniques adéquates qui sont capables de guider et de promouvoir l'ingénierie simple, flexible, rapide et unifiée des services fondés sur la sémantique. En réalité, les plates-formes actuelles des services web sémantiques ne prennent pas vraiment en charge la construction des services web sémantiques "*intelligents et auto-composables*" et elles demeurent limitées à de simples annotations sémantiques des services web ou à des intégrations et des utilisations rudimentaires de quelques concepts associés à des ontologies [82].

2.7.3 Le gestion des contraintes fonctionnelles et non fonctionnelles

Bien que beaucoup de standards ont émergés durant la dernière décennie pour la spécification des techniques permettant la prise en compte des différents phases du cycle de vie des services web, la majorités des approches proposées se focalise en grande partie sur les contraintes fonctionnelles des services web (*publication, découverte, sélection, interaction, ...*). Par contre, les contraintes non fonctionnelles, telles que les paramètres de qualité des services *QoS* [83, 84, 85] (*performance, cout, fiabilité,...*), ou la sécurité et l'administration des services, n'ont pas bénéficié de tout l'intérêt qu'elles revêtent. En effet, bien que ce type de contraintes exprime les garanties et les conditions auxquelles le client doit se confirmer durant les différentes phases du processus d'interaction avec le service désiré (*découverte, sélection, invocation, composition*), peu de standards ont été proposés pour leur prise en charge, et même ceux proposés n'ont pas encore atteint un niveau de maturité qui les rend opérationnels de façon effective.

A titre d'illustration, il est constaté une absence de modèles standards pour la spécification des métriques de qualité de service, et même ceux qui sont en cours de gestation ne sont pas encore supportés par la structure actuelle du registre UDDI. Comme conséquence directe, une grande partie des travaux de recherche autours de cette problématique [86, 87, 88, 89] convergent vers des propositions de remaniements et des extensions de cet annuaire afin de permettre la prise en compte des contraintes non fonctionnelles.

2.7.4 La représentation des protocoles et les transformations de modèles

Au niveau conceptuel, différents modèles dotés de niveaux d'expressivité variés ont été proposés dans la littérature pour représenter les protocoles des services web. Certains sont formels [10, 11, 52, 65], alors que d'autres sont plutôt graphiques. Dans cette sphère, certaines approches proposent même des techniques de rétro-engineering pour rehausser le niveau d'abstraction, en transformant des programmes écrit dans des langages spécifiques vers des modèles plus abstraits [90, 91]. Vue la différence du niveau d'expressivité des différents modèles proposés, et pour tenir compte des besoins spécifiques des utilisateurs qui se sont familiarisés avec une catégorie donnée de modèles, toute une panoplie de techniques de transformation entre les différents modèles (*d'un modèle source vers un autre modèle cible*) a été proposée dans la littérature de recherche et des outils de passage inter-modèles ont été élaborés (*voir par exemples [92, 93, 94]*).

Nous estimons que les modèles formels (*e.g : automates, réseaux de Petri*) restent les plus adéquats pour une représentation efficace des protocoles des services web, puisqu'ils sont dotés d'une assise théorique solide. En plus, ils peuvent être soumis à des techniques de vérification de modèles (*model-checking*) [95].

2.7.5 Evolution, adaptation et flexibilité des services web

Dans les environnements service web, les applications et les systèmes d'informations peuvent être construits en intégrant différents services hétérogènes provenant de divers fournisseurs. Cependant, comme les services utilisés sont assujettis aux changements qui sont dûs à l'évolution des lois et réglementations, la gestion de l'évolution des services et l'analyse de l'impact du changement sont des questions d'actualité qui ont fait l'objet de travaux de recherche très intenses. Ainsi, pour faire face à cet enjeu qui caractérise les systèmes logiciels plusieurs approches ont été proposées, et chacune se focalise sur un aspect particulier du problème, entre autres : l'actualisation de l'interface du service, les changements dans les paramètres non fonctionnels et l'évolution du comportement des protocoles du service [96, 97, 98]. Cette dernière question sera abordée en détails dans les prochains chapitres (*Chap. 3 et 4*).

2.8 Conclusion

Dans ce chapitre, nous avons présenté le principe de fonctionnement des environnements services web. Nous avons exposé les protocoles standards qui leur sont associés, et nous avons illustré l'intérêt de la technique de leur composition. L'accent a été mis particulièrement sur l'importance de la description de leurs comportement externes par des protocoles de services qui emploient divers modèles de représentation. La chapitre a été clôturé par des discussions sur les difficultés qui entravent le déploiement effectif et à grande échelle des services web. Nous avons aussi soulevé quelques défis auxquels cette technologie est encore confrontée.

Le prochain chapitre sera consacré aux processus métiers, à la gestion de leurs évolutions et à l'analyse de l'impact des changements.

Chapitre 3

Les processus métiers et leurs évolutions

3.1 Introduction

Ce chapitre est dédié aux processus métiers et à la gestion des évolutions induites par l'application des opérations de changements.

Dans la première section (*section 3.2*), nous introduisons les définitions et les concepts de base associés aux processus métiers et aux systèmes assurant leur gestion. Dans la section 3.3, nous abordons les raisons et les types de changements. La section 3.4 met en exergue l'intérêt de l'analyse de l'impact des évolutions des processus métiers et ses différentes dimensions. Dans la section 3.5, nous exposons les approches de flexibilité pour la gestion des changements et, puis, la section 3.6 traite des techniques de migration des instances actives d'un processus métier qui a subit des évolutions. Les choix retenus pour la poursuite de nos travaux sont explicités dans la section 3.7, et nous terminerons le chapitre par une conclusion en section 3.8 qui récapitule les enjeux inhérents aux évolutions des processus métiers.

3.2 Notions de base sur les processus métiers

Aujourd'hui, la plupart des activités économiques et sociales sont collaboratives, traversent les frontières géographiques des organisations et sont opérées dans des environnements fortement dynamiques et ouverts. L'accomplissement de telles activités passe, inévitablement, par la maitrise et la réalisation de plusieurs processus métiers complexes qui sont gérés par les différents acteurs socio-économiques impliqués dans le domaine de gestion en question.

Dans ce qui suit, nous nous focalisons sur le concept de processus métier et sa jonction avec la technologie des services web.

3.2.1 Processus métiers et instances de processus

Généralement, un processus métier ou "**business process**" est défini par :

Définition 3.1 *Un processus métier est un ensemble d'activités exécutées de manière coordonnée dans un environnement organisationnel et technique pour réaliser un objectif métier particulier [99, 100].*

Une définition plus précise d'un processus métier est donnée par le WfMC [1]

1. Workflow Management Coalition

Définition 3.2 *Un processus métier est un ensemble de procédures ou d'activités liées les unes aux autres pour atteindre collectivement un objectif métier en définissant les rôles et les interactions fonctionnelles au sein d'une structure organisationnelle.*

Cette définition met en exergue une collection d'activités et de tâches organisées dans le temps qui, une fois achevées, permettront d'atteindre un objectif organisationnel et un résultat précis.

D'un point de vue organisationnel, un processus métier est considéré comme un ensemble de relations logiques entre un groupe d'activités impliquant différents partenaires et participants, tels que : des services du S.I ou des applications, des acteurs humains ou d'autres processus métiers. Du point de vue informationnel, le processus métiers est considéré comme une forme d'échanges d'informations et de données entre les acteurs concernés. Cet échange permet de fournir une valeur ajoutée tangible aux clients et aux différents interlocuteurs de l'organisation.

Le méta-modèle proposé par Morley et al [2] illustre les interactions entre les différents concepts précédents.

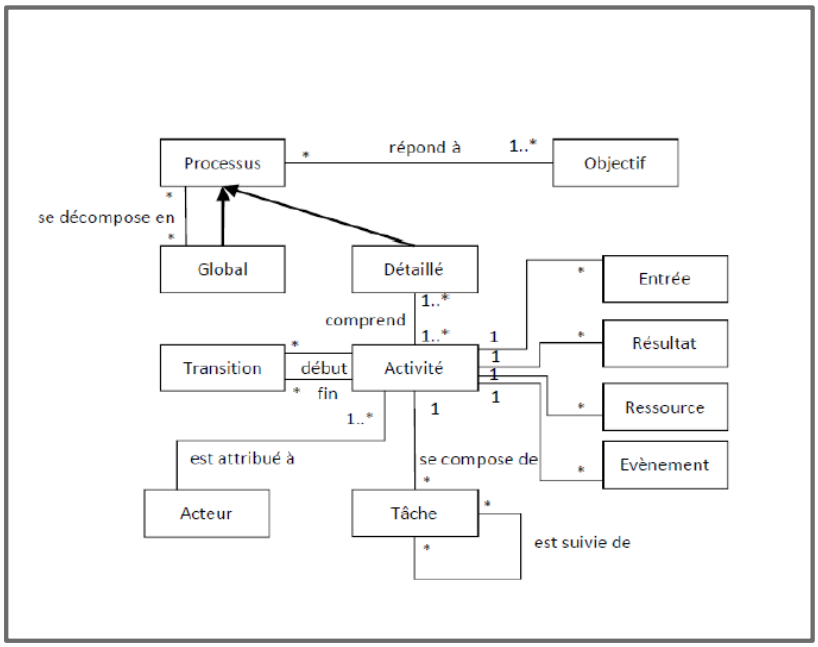

FIGURE 3.1 – Méta modèle du processus métier selon Morley et al [2]

Un aspect fondamental sur lequel repose les processus métiers est leur représentation. Il prendra en compte les activités explicites et les contraintes d'exécution qui leur sont imposées. Dans cette perspective, des modèles (*ou schémas*) sont, souvent, utilisés pour exprimer de telles représentations. Les représentations graphiques sont très adaptées pour exprimer les contraintes qui régissent l'exécution d'un processus métier ; par exemple, l'ordre d'exécution des activités appartenant au processus métier particulier. D'autre part, l'exécution d'un processus métier, conformément à son modèle, génère *des instances d'exécution du processus*, que nous définissons ci-après.

Définition 3.3 *Une instance d'exécution d'un processus métier représente un cas réel d'un processus opérationnel d'une compagnie, et consiste en des instances structurées d'activités pour le processus en question [99].*

Il en résulte que chaque modèle du processus métier agit comme un plan pour l'ensemble des instances du processus métier et que chaque activité agit comme un plan pour l'ensemble des instances des activités.

3.2.2 La gestion des processus métiers

En sa qualité de composant central sur lequel s'articule toute la gestion d'une organisation, un processus métier exige d'être géré, suivi et supervisé. La gestion des processus métiers ou "**B**usiness **P**rocess **M**anagement" (**BPM**) est la fonction de l'organisation qui assure la prise en charge des différentes phases relatives au cycle de vie des processus métiers (*voir section 3.2.3*). Elle est définie par [3], comme suit :

Définition 3.4 *La gestion des processus métiers (**BPM**) est l'utilisation de méthodes, techniques et systèmes logiciels pour concevoir, exécuter, contrôler et analyser des processus opérationnels faisant intervenir des hommes, des applications, des documents et d'autres sources d'information.*

De ce point de vue, la gestion des processus métiers **(BPM)** est considérée comme une approche de management, axée sur l'alignement continue de tous les aspects d'une organisation avec les besoins réels des clients. Elle vise à placer les processus métiers au centre d'une réflexion globale d'intégration, ou sous le concept de "**process-centric**". En effet, le but est de favoriser l'efficacité opérationnelle, tout en abordant la question de l'évolution continue des processus métiers et en avantageant le point de vue "*métier*" sur le point de vue "*technique*". Cette façon de voir l'entreprise rend les processus plus efficaces et plus capables de s'adapter aux éventuels changements de l'environnement.

En définitive, la gestion des processus métiers est une approche qui favorise la perception de l'organisation en tant que système composé de processus métiers interconnectés. Cette orientation guide toute l'organisation afin de s'assurer que ses processus métiers sont mis en œuvre efficacement, tout en répondant aux besoins de ses différents interlocuteurs, et avec un niveau de performances optimal aussi bien qu'avec une bonne maîtrise de ses coûts [99].

Une fois les processus métiers définis et modélisés, ils sont soumis à l'adoption par les différents partenaires qui vont les exécuter. Aussi, ils peuvent être soumis à des actions d'analyse des performances et à des opérations de maintenance et d'actualisation, en vue de les améliorer. Traditionnellement, les processus métiers sont exécutés manuellement et en conformité avec les règles métiers de l'entreprise. Actuellement, les processus métiers tirent profit des avancées technologiques et sont pris en charge par des logiciels spécifiques, appelés : les systèmes de gestion de processus métiers, ou Business Processes Management Systems (**BPMS**), définis ci-dessous.

Définition 3.5 *Un système de gestion de processus métiers est un logiciel générique utilisé pour coordonner l'exécution des activités de l'organisation. Il est guidé par les représentations explicites des processus métiers. [99]*

3.2.3 Cycle de vie d'un processus métier

Dans les environnements organisationnels axés sur les processus métiers et en adéquation avec la démarche **BPM**, la gestion du cycle de vie d'un processus métier vise à assurer l'accompagnement du gestionnaire durant toutes les phases, depuis la conception du processus, au pilotage, tout en traitant en permanence les évolutions selon les objectifs métiers et les contraintes qui surgissent dans l'environnement de l'organisation. Afin d'atteindre de tels objectifs, la mise en œuvre de niveaux d'abstraction, aussi bien *théoriques* que *pratiques* pour les processus métiers, à travers plusieurs spécifications et différentes perspectives, est incontournable.

Dans la littérature, il n'y a pas de vue uniforme sur le nombre de phases du cycle de vie. Ce nombre varie en fonction de la granularité choisie pour l'identification des phases, mais généralement on distingue les quatre phases illustrées dans la figure 3.2.

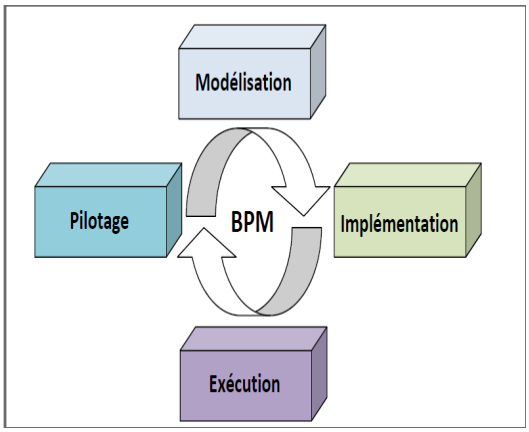

FIGURE 3.2 – Cycle de vie d'un processus métier (*Wil. Van Der Aalst. et al* [3])

- **la phase de modélisation** : durant cette phase, les experts métiers définissent d'une manière détaillée les processus métiers à l'aide d'outils de modélisation graphiques en adoptant, généralement, le standard BPMN [67]. Ils spécifient l'enchainement des tâches du processus métier. En raison du manque d'informations techniques (*gestion des exceptions, formats de données*) pour les activités, le modèle conçu doit être transformé en un modèle de processus exécutable.
- **la phase d'implémentation** : le processus créé dans la phase de modélisation est enrichi dans le but d'être exécuté par un moteur de processus métiers (*Business Process Engine :* **BPE**). A cet effet, le langage standard BPEL [54] est utilisé pour spécifier le déroulement du processus et pour exprimer les séquences d'évènements du processus métier. Le modèle de processus exécutable obtenu peut, à présent, être déployé et pris en charge par un moteur de processus métiers qui mettra en œuvre les règles métiers décrites.
- **la phase d'exécution** : le processus exécutable est interprété par le moteur de processus BPE. Le moteur BPE est le responsable des interactions entre les éléments du processus (*les documents, les informations et les tâches*). Il exécute les instances de processus, tout en déléguant les tâches automatiques aux services web et les tâches manuelles aux acteurs.

– **la phase de pilotage et d'optimisation** (*Business Activity Monitoring :* **BAM**) : Les informations disponibles sont exploitées pour évaluer et optimiser les modèles de processus et leur implémentation. Les journaux logs sont traités pour s'assurer que les performances ne se dégradent pas au fil du temps, pour améliorer l'efficacité des processus ou encore pour contrôler le bon déroulement de l'activité à travers l'analyse des tableaux de bord et en exploitant des indicateurs de performance.

3.2.4 Implémentation des processus métiers

Au cours des dernières années, le phénomène de la globalisation conjugué aux progrès réalisés dans le domaine des technologies de l'information et de la communication, en particulier les technologies du web, ont modifié profondément et de manière spectaculaire la façon de gérer les processus métier au niveau des organisations. En particulier, dans l'environnement Internet actuel, les services web constituent un choix technologique avantageux pour l'implémentation des processus métiers transversaux des entreprises. Ainsi, il est fréquent d'observer qu'aujourd'hui dans les environnements de plus en plus ouverts et versatiles des organisations, les processus métiers sont implémentés en tant qu'applications publiées sur Internet comme des services web. Ces applications peuvent être invoquées par différents partenaires via des protocoles standards. L'acquis immédiat de ces avancés est que l'intégration des applications intra et inter-entreprises est considérablement facilitée et l'interopérabilité entre les différents partenaires est plus transparente.

La figure 3.3, inspirée de [99], met en relief les composants de l'architecture d'un environnement d'exécution des processus métier, les relations entre les différents composants et leur jonction avec la technologie des services web.

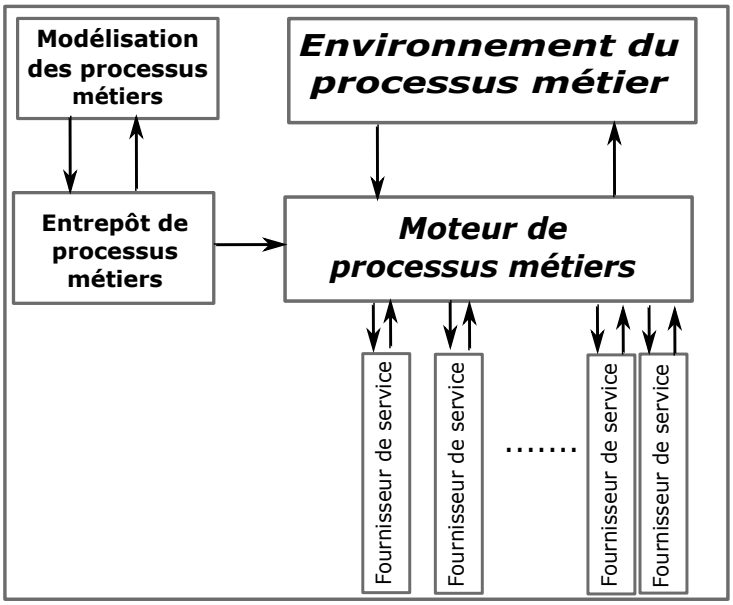

FIGURE 3.3 – Jonction entre système de gestion des processus métiers et services web

Le composant modélisation des processus métier offre des outils pour la création

des modèles des processus métiers, en définissant leurs structures et les informations sur les activités. Les processus métiers créés sont stockés dans un entrepôt. Enfin, l'environnement des processus métiers (*clients, autres processus métiers, services web*) déclenche l'instanciation et l'exécution des processus métiers qui seront pris en charge par le moteur d'exécution. Ce dernier peut interagir avec d'autres fournisseurs de services qui réalisent des activités du processus métier en cours d'exécution. Ces activités peuvent être des services web ou, simplement un savoir faire particulier détenu par des acteurs humains [99].

Dans un scénario opérationnel typique offrant l'implémentation des processus métiers sous forme de services web, le système de gestion des processus métiers (**BPMS**) exécute les instances des processus, tout en déléguant les tâches automatiques aux services web et les tâches manuelles aux acteurs. Lors de la production d'une exception durant l'exécution du processus, le moteur des processus métiers (**BPE**) lance une action de compensation pour amener le processus à une exécution antérieure valide.

De ce point de vue technologique, les organisations modernes peuvent être considérées comme des systèmes ouverts étroitement liés à un environnement très versatile. Dans un tel environnement très assujettis aux changements, la survie de l'organisation dépend, souvent, de sa maitrise de ses processus métiers et de sa capacité d'adaptation aux évolutions.

Pour mettre la lumière sur cet enjeu stratégique auquel sont confrontées les entreprises modernes, nous consacrons les prochaines sections de ce chapitre aux aspects relatifs aux changements, aux évolutions et à l'analyse d'impact.

3.3 Gestion des changements des processus métiers

L'évolution d'un processus métier résulte directement de l'application d'un ou de plusieurs changements (*par exemple l'ajout ou la suppression d'une activité*) sur les éléments descriptifs du processus (*activités, données, acteurs, ...*). Ces changement peuvent être motivés par des raisons très variées. Dans les travaux de [101], les changement ont été catégorisés dans les quatre classes suivantes :

- **Les changements évolutifs** : permettent de prendre en charge de nouveaux besoins, ou bien ils sont opérés pour s'adapter aux nouvelles lois et réglementations dictées par l'environnement de l'organisation.
- **Les changements structurels** : sont dûs à des modifications dans la structure d'une organisation, telles que les rachats et les fusions d'entreprises. Dans certaines situations, ils sont occasionnés par des changements de la nature de l'activité de l'entreprise qui induisent sa restructuration et une remise en ordre de son mode de fonctionnement.
- **Les changements perfectifs** : ces changements sont justifiés par des besoins d'optimisation des processus métiers dans le cadre de l'amélioration des critères de performances, particulièrement : un besoin de gain de temps d'exécution, un souci de réduction du temps de réponse pour les clients, l'optimisation des procédures et des ressources, ...
- **Les changements correctifs** : sont introduits en réponse à la constatation de certaines anomalies (*erreurs ou exceptions*) durant l'exécution des processus métiers.

Comme réponse immédiate à l'apparition des motifs de changement sus-cités, les entreprises doivent continuellement mettre à jour et améliorer leurs processus métiers qui sont publiés sur internet en tant que services web, afin de s'adapter aux changements qui se produisent à un rythme de plus en plus fréquent. Cette exigence de flexibilité des processus métier suscite depuis plus de vingt ans un intérêt croissant aussi bien de la part des industriels des logiciels que de la part de la communauté scientifique. Le défi à surmonter est d'offrir des outils et des mécanismes permettant l'intégration et la gestion des changements permanents des processus métier d'une manière flexible et efficace.

Dans un contexte d'évolution des processus métiers, l'enjeu fondamental est de répondre aux soucis des gestionnaires de protocoles et qui se résument, essentiellement, dans les préoccupations de type :

- Quel est l'impact des changements sur les instances en cours d'exécution ?
- Que faire des instances en cours d'exécution ? Quel modèle de processus métier vont-elles suivre (*l'ancien ou le nouveau*) ?
- Comment propager les changements sur les instances en cours d'exécution ?
- Comment garantir la cohérence structurelle et comportementale du processus métier ayant subi des modifications ?
- Peut-on spécifier des stratégies de migration spécifiques à des besoins particuliers, suivant l'état de progression de chaque instance, par exemple ?

La réponse aux question précédente passe, impérativement, par la conception d'une approche globale pour la gestion des changements des processus métiers. Cette dernière exige, inévitablement, une compréhension approfondie de la nature et des types des changements à appliquer sur les modèles de processus métiers.

Dans ce qui suit, nous commençons par détailler la typologie des opérations de changement appliquées aux processus métier, puis nous discutons de la portée des changements et enfin, nous exposons les propriétés inhérentes aux changements.

3.3.1 La typologie des opérations de changement

Les opérations de changement qui affectent un processus métier peuvent être basiques ou complexes.

a) Les opérations basiques de changement :

Les opérations basiques permettent d'insérer, de supprimer ou de modifier une simple activité, changer un flux de contrôle ou un flux de données ou encore, changer l'affectation d'un rôle associé à un acteur donné.

Les deux figures 3.4 et 3.5 ci-dessous montrent des exemples d'opérations basiques de changement opérées sur des modèles de processus métiers.

Les automates d'états finis déterministes (**AFD**) sont utilisés comme modèle de représentation des processus métiers (*ce choix sera motivé dans la chapitre 5*, section 5.2). Les lettres de l'alphabet (a,b,...,z) désignent les activités des processus, alors que S et S' dénomment, respectivement, l'ancienne et la nouvelle version du processus métier.

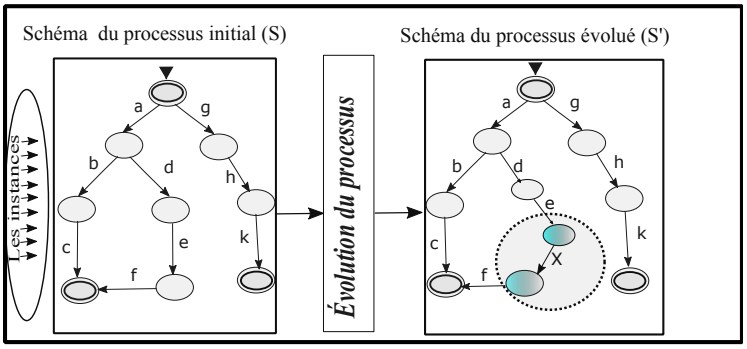

FIGURE 3.4 – Changements basiques : Insertion d'une activité et d'un état

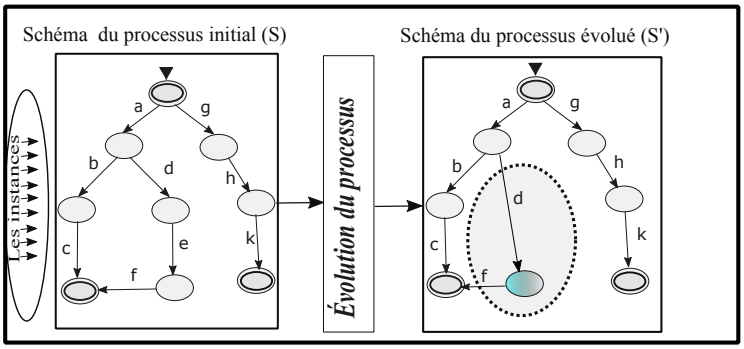

FIGURE 3.5 – Changements basiques : Suppression d'une activité et d'un état

Après chaque changement, l'ancienne version du modèle de processus (*à gauche de la figure*) et sa version évoluée (*à droite de la figure*) sont exposées et les modifications sont mises en évidence dans des zones ombrées et encerclées en pointillés.

La figure 3.4 illustre un exemple d'opérations basiques qui consiste à insérer un état et une activité X entre les deux activités e et f.

Dans la figure 3.5, les changements opérés sur le schéma du processus métier S consistent, plutôt, à supprimer un état et l'activité e qui se trouve entre les deux activités d et f.

b) Les opérations complexes de changement :

Les opérations complexes résultent de la combinaison des opérations de changement basiques précédentes et agissent sur des fragments de processus (*séquence d'activités*). Elles consistent à remplacer un ou plusieurs fragments de processus par d'autres qui sont plus actualisés. Parfois, ces opérations peuvent supprimer ou à ajouter des fragments de processus ou opérer d'autres actions complexes, telles que le déplacement d'un fragment d'un processus d'une position Pos_1 vers une autre position Pos_2 (*suppression du fragment de la position* Pos_1 *et son insertion à la position* Pos_2). Dans d'autres circonstances, leur action permet de permuter ou de paralléliser des fragments de processus.

Quelques exemple d'opérations de changement complexes sont présentées dans les deux figures 3.6 et 3.7, ci-dessous.

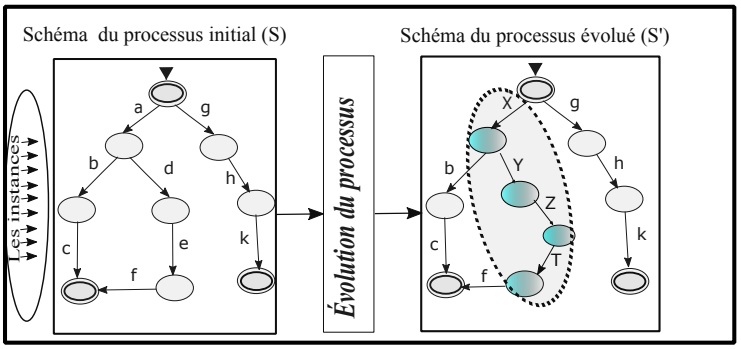

FIGURE 3.6 – Changements complexes : Substitution d'un fragment de processus

Dans la figure 3.6, l'activité (a) a subi une opération de remplacement simple (*l'activité (a) est substituée par l'activité X*). En même temps, le fragment (d.e) est remplacé par un nouveau fragment (Y.Z.T). Suite à l'application de ces changements, dans le processus évolué S' l'ancien chemin (*a.b.c)* est remplacé par le chemin (X.b.c) et le chemin (a.d.e) est actualisé par la prise en charge du nouveau chemin (X.Y.Z.T). Dans la figure 3.7, le fragment (b.c) est déplacé vers une autre position pour exprimer

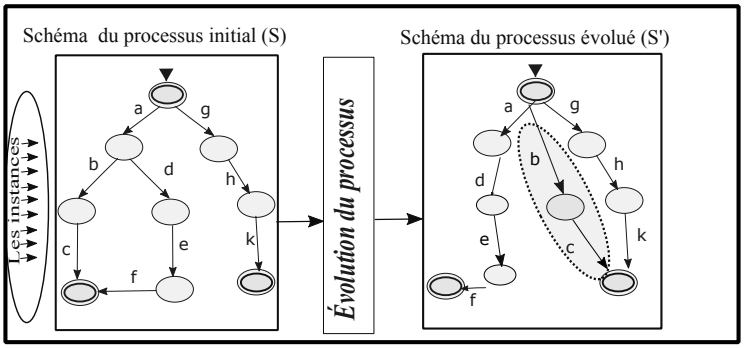

FIGURE 3.7 – Changements complexes : Déplacement d'un fragment de processus

de nouvelles règles de gestion.

D'autres patrons de changement complexes sont détaillés dans les travaux de Weber et al [14].

3.3.2 La portée des changements des processus métiers

Abstraction faite aux raisons qui induisent les évolutions des processus métiers, les changements qui ont engendré cette évolution peuvent porter sur deux niveaux distincts :

a) Le niveau des instances du processus métier.

b) Le niveau du schéma du processus métier.

a) Changements au niveau instances

L'application des changements au niveau des instances permet de faire face à des situations imprévues ou à des exceptions qui se manifestent lors de l'exécution d'un processus métier [102]. Leur mise en œuvre porte uniquement sur des sous ensembles spécifiques d'instances qui sont ciblées au cas par cas. Ces changement sont souvent appliqués d'une manière "ad-hoc".

Par exemple, en se référant au schéma de la figure 3.6, une instance donnée (i) ayant commencé son exécution sur la base de l'ancien schéma du processus (S), peut être modifiée individuellement. En se sens, si cette instance a exécuté la séquence d'activités : a et b, alors elle sera rendue conforme avec le nouveau schéma du processus (S') en substituant l'activité a par l'activité X. la nouvelle instance continuera son exécution dans le nouveau processus métier (S') sur la base de la nouvelle séquence d'activités : X suivie de **b** au lieu de a suivie de **b**.

Généralement, les changements au niveau instances ne génèrent qu'un impact local et ils n'ont pas d'incidences sur le reste des instances du processus métier en cours d'exécution [103].

b) Changements du schéma du processus métier

En général, agir sur les instances en cours d'exécution n'est pas suffisant pour résorber toutes les difficultés qui surgissent suite à l'application des changements. Par ailleurs, cette façon de gérer les effets des évolutions s'avère très lourde, notamment quand le nombre d'instances affectées par les changements est considérable. Plutôt, il serai plus judicieux d'agir sur le schéma du processus métier ; c'est la structure du processus métier elle-même qui doit être remise en question [104].

L'évolution du schéma d'un processus métier nécessite souvent la propagation des changements sur le reste des éléments du schéma et sur les instances en cours d'exécution. En effet, ce type de changement produit un impact global [103].

La figure 3.6, illustre un changement au niveau du schéma du processus métier. Le schéma (S') est le résultat des changements apportés sur le schéma initial (S) et qui consistent à substituer un fragment du processus par un autre (*le chemin (a.d.e) dans (S) est remplacé par le chemin (X.Y.Z.T) dans (S')*). Dans ce scénario d'évolution, toutes les instances ayant commencé leur exécution dans l'ancien processus (S) par l'invocation de l'activité (a) seront affectées par le changement du schéma du processus. D'une manière générale, des actions spécifiques et ciblées doivent être opérées afin de rendre les instances conformes au nouveau schéma (S').

Abstraction faite au nombre d'instances en cours d'exécution, l'application des changements au niveau des instances s'avère plus simple à gérer, car elle tient compte uniquement de l'état réel de progression de chaque instance (*traitement au cas par cas*). Par contre, des difficultés surgissent lors de l'application des changements au niveau schéma. En effet, dans ce dernier cas les changements ne peuvent être accomplis sans quelques difficultés liées aux instances en cours d'exécution qui ne sont plus conformes à la nouvelle version du schéma. A titre d'illustration, une instance (i1), ayant emprunté le chemin (g.h.k) de la figure 3.6 peut migrer vers le schéma (S'), contrairement à une instance i2 qui a emprunté un chemin commençant par l'activité

(a). Cette instance doit donc être réalisée sur la base de la version initiale (S) du schéma du processus.

De ce qui précède, il ressort qu'une analyse de l'impact des changements doit être élaborée par le gestionnaire de protocole afin de déterminer les instances qui peuvent migrer vers la nouvelle version de processus, tout en spécifiant leurs propriétés (*état d'exécution, la séquence correspondante d'activités, ...*) dans le processus évolué.

Cet aspect sera abordé dans la section 3.4

3.3.3 Les propriétés des changements

Selon Regev et al [105], les changements dans les processus métiers sont caractérisés par les quatre propriétés suivantes :

a) L'ampleur du changement

Deux options sont offertes aux gestionnaires de protocoles pour la mise en œuvre des changements qu'ils désirent effectuer. Si les modifications à faire sont minimes, ils optent pour un changements incrémental, cependant ils sont souvent contraints de faire des changements révolutionnaires.

Un changement révolutionnaire aboli le schéma du processus existant et crée un nouveau schéma tandis qu'un changement incrémental apporte des modifications progressives qui s'appliquent sur le schéma d'un processus métier existant.

b) La durée du changement

Les changements peuvent être temporaires ou permanents. Un changement temporaire n'est valable que pour une période limitée, ce qui implique la possibilité de retour au schéma de départ après la fin d'une échéance précise. Un changement permanent demeure valable jusqu'à ce qu'un changement futur se manifeste. Dans ce dernier cas, on transite de version en version pour le même processus métier.

c) La rapidité du changement

Un changement peut être immédiatement pris en compte ou être différé pour un moment opportun et précis. Un changement immédiat répond, généralement, à une situation d'urgence et il est appliqué à toutes les instances du processus, y compris celles qui sont en cours d'exécution. Cela implique qu'**un mécanisme de migration** des instances doit être actionné. Un changement différé n'est appliqué qu'aux nouvelles instances du processus en question et ses instances en cours d'exécution restent inchangées. Cela implique, qu'à un instant donné il y a coexistence de différentes versions d'un même schéma de processus.

d) L'anticipation du changement

L'anticipation du changement est une préoccupation qui vise à savoir si un changement est planifié ou ad-hoc. Un changement planifié ou prévu exprime, généralement,

une refonte temporaire ou permanente d'un processus tandis qu'un changement ad-hoc est souvent appliqué pour faire face à des situations exceptionnelles qui surgissent durant le cycle de vie d'un processus métier.

3.4 Analyse de l'impact du changement

La mise en application concrète des opérations de changements n'est pas sans incidences sur le contexte global du processus métier et ses divers éléments. Certainement, quelque soient les raisons ayant causé les changements et quelque soient les types des opérations de changements (*basiques vs complexes*), toute action de changement peut agir de manière directe, aussi bien sur le schéma que sur les instances du processus métier en question. En effet, d'une part, l'évolution engendrée par les changements influencera sur les autres structures de la spécification du processus métier concerné, et d'autre part ces changements vont se répercuter sur la progression des instances en cours d'exécution.

Le terme générique **analyse de l'impact du changement** est, généralement, employé par les spécialistes pour désigner les approches et les techniques qui abordent les questions relatives aux conséquences des changements sur les processus métiers.

Dans cette section nous nous focalisons sur cette dimension inhérente aux évolutions des processus métiers, nous définissons le concept d'impact du changement, et nous mettons en relief le besoin d'une analyse de son impact.

3.4.1 Notion d'impact du changement

Le concept d'**impact du changement** n'est pas spécifique au domaine des processus métiers. En effet, force est de constater que les notions de changement et d'évolution sont des aspects intrinsèques à plusieurs disciplines de l'informatique qui sont touchées par ce phénomène. Bien que divers domaines tels que : les bases de données, le génie-logiciel ou encore les ontologies ont été confrontés aux problématiques associés aux changements, les processus métiers et les workflows demeurent les deux domaines qui ont largement étudié et exploré la question du changement et son impact.

Une définition explicite de l'impact du changement est donnée par [101].
"*Le concept* ***impact du changement*** *fait référence aux répercussions et aux effets qu'un changement d'une entité du système peut avoir sur les autres entités de ce dernier.*"

Pour cerner, comprendre et gérer les divers effets de vague induits par l'impact de changement, le concept : **analyse de l'impact du changement** est utilisé. Nous définissons ce concept dans ce qui suit.

3.4.2 Analyse de l'impact du changement

Dans la littérature, différents définitions ont été données à la notion d'analyse de l'impact des changements. Parmi ces définitions, nous pouvons retenir celle de Haney et al [106] qui porte sur une technique d'analyse de la connexion des différents modules d'un système. Dans [101], l'analyse de l'impact du changement est vue comme une

technique qui a pour objectifs d'évaluer les effets générés après l'application des changements. Pfleeger et al [107] définissent l'analyse de l'impact du changement comme étant :" *l'évaluation des nombreux risques associés au changement, y compris les estimations des effets sur les ressources, sur les efforts et sur la planification".*

Une définition plus pertinente de l'analyse de l'impact du changement est donnée dans [108]. Elle stipule qu'il s'agit de :

Définition 3.6 *L'identification des conséquences potentielles d'un changement, ou l'estimation des besoins à prendre en charge pour accomplir un changement, tout en se concentrant sur la portée des changements et les détails conceptuels induits.*

Cette dernière définition stipule que l'analyse de l'impact du changement désigne toute approche et/ou mécanisme permettant d'identifier les conséquences potentielles d'un changement, ou l'estimation de ce qui doit être modifié pour réaliser un changement.

La technique d'analyse d'impact du changement exposée dans [106] est basée sur l'idée que pour chaque paire de modules dans un système logiciel, il existe une probabilité qu'un changement dans un module nécessite un autre changement conséquent dans d'autres modules. La technique a ensuite été utilisée pour la propagation du changement de modèles entre tous les composants du système logiciel.

Plus directement liés à nos préoccupations, dans [101] un changement de processus métier est décrit formellement comme une différence (*notée* $\triangle$) entre le schéma du processus métier initial (S) et le schéma du processus métier (S') qui résulte de l'application des changements. Cette évolution (*variation*) est quantifiée par : $(S') = \triangle + (S)$, soit : $\triangle = (S') - (S)$. La variation $\triangle$ peut générer différents types d'impact qui peuvent être structurels, fonctionnels, comportementaux, logiques et/ou qualitatifs sur une partie ou sur l'ensemble du modèle de processus métier et ses instances. Ces impacts peuvent être propagés d'une manière horizontale (*intra-couche*) et verticale (*inter-couches*).

Pour illustrer le concept d'analyse d'impact du changement, nous citons à titre d'exemple l'analyse exposée dans [101], où l'auteur propose de vérifier la cohérence et de la conformité des modèles de processus métier, après chaque changement mais aussi, d'établir une évaluation *a priori* de l'impact structurel et qualitatif des modifications. Dans cette étude, l'analyse de l'impact est basée sur la relation générique dite de *dépendance*. En effet, une telle relation permet de dire : *si une entité* B *dépend d'une autre entité* A *alors le changement de* A *aura un impact sur* B.

Dans la littérature il existe toute une variété d'approches et de techniques permettant la prise en charge et l'intégration des changements des processus métiers. En tenant compte du niveau de préoccupation pour la mise en œuvre des changements (*schéma vs instance*), nous pouvons classer ces techniques en deux grandes catégories : les techniques de **flexibilité** qui agissent sur le niveau schéma du processus métier et les techniques de **migration** qui sont utilisées pour gérer les instances du processus.

La suite du chapitre est consacrée à l'exposé de ces deux techniques.

3.5 Les techniques de flexibilité

Pour faire face aux défis inhérents aux évolutions des processus métiers, les entreprises doivent être en mesure de s'adapter continuellement aux changements qui

surgissent dans leur environnement ; i.e. elles doivent être **flexibles**. La flexibilité des entreprises réside dans leur capacité à contrôler l'évolution de leurs processus métiers en étant en mesure, à la fois, de mettre en place de manière efficace et peu coûteuse de nouveaux processus métiers, et aussi d'adapter ceux déjà existants.

Plusieurs concepts ont été associés à de telles capacités d'accommodation des entreprises, tels que : la résilience stratégique, l'agilité et l'adaptation.

Dans [109], *la résilience stratégique*, définie comme *la capacité des entreprises à réinventer* ***dynamiquement*** *les modèles de processus métiers et les stratégies adéquates en fonction des circonstances*, a été identifiée comme un facteur clé pour soutenir la compétitivité et la croissance des entreprises modernes. Pour atteindre ces objectifs, la notion d'*entreprise agile* a émergé en tant qu'ensemble de concepts, d'architectures, de méthode et de technologies permettant de soutenir les besoins croissants des organisations pour l'adaptation rapide et efficace aux conditions très variables de leur environnement. Un des piliers de l'agilité des entreprises réside dans leur capacité à contrôler la dynamique de leurs processus métiers tout en étant en mesure, à la fois, de mettre en place de manière efficace et peu coûteuse de nouveaux processus métiers et aussi, d'adapter les processus existants [110].

Le concept de *flexibilité* des processus métiers est utilisé pour désigner de telles capacités d'adaptation. Généralement, la flexibilité est définie comme étant la capacité de s'accommoder aux changements sans disparaître, à savoir sans perdre d'identité.

Un processus métier est dit *flexible* s'il est possible de le changer sans le remplacer complètement. C'est à dire que les changements sont pris en charge dans le processus métiers lui même (*au niveau schéma et/ou au niveau de ses instances*) par la mise à jour des seuls éléments concernés par les modifications, et tout en préservant les autres parties stables.

Deux approches de flexibilité ont été distinguées dans [101] : l'approche par sélection à postériori "*Late binding*" et l'approche par modélisation à postériori "*Late modeling*".

3.5.1 l'approche par sélection à postériori

Dans cette approche, les différents éléments du processus métier sont spécifiés préalablement lors de la phase de conception, mais ils sont considérés comme des objets dont le comportement est défini lors de la phase d'exécution [101]. Cette façon d'énumérer exhaustivement les différents comportements possibles implique qu'aucun changement n'est nécessaire sur le schéma du processus métier, mais un large choix d'alternatives d'exécution est disponible durant cette phase [111]. Ceci inclut la capacité de sélection de la meilleure ressource ou de l'acteur le plus approprié pour réaliser une activité qui doit satisfaire un critère donné.

Un exemple de mise en œuvre de ce type d'approches est présenté dans [112]. Dans cette approche, l'outil appelé "AFLOWS" offre la possibilité d'associer la même définition d'un processus métier pour une large variété d'occurrences (*instances*) dont les exécutions seront en fonction du contexte de chacune.

3.5.2 l'approche par modélisation à postériori

Contrairement à l'approche précédente, cette approche laisse certaines parties du processus métier ouvertes à l'innovation et à la créativité des utilisateurs. En particulier, lorsque certaines spécifications ne peuvent pas être identifiées à priori lors de la phase de modélisation, l'utilisateur pourra les spécifier d'une manière dynamique à l'exécution. En ce sens, certaines activités du processus peuvent être déclarées comme critiques et obligatoires, alors que d'autres pourraient être optionnelles, et ne seront prises en compte que lors de la phase d'exécution. Cette façon de percevoir les processus métiers donne la possibilité aux utilisateurs de personnaliser les processus métiers à leur besoins spécifiques [101].

Un exemple illustratif de ce type d'approches est exposé dans [113], où différents comportements peuvent être associés aux activités du processus métier. Dans ce cas, l'utilisateur pourra sélectionner le comportement le plus approprié dynamiquement (*à l'exécution*). Dans la même optique, les travaux de [114] offrent à l'utilisateur la possibilité de sélectionner les performances les plus appropriées pour une activité qui a été initialement définie comme un objectif à atteindre. En plus, si aucune méthode existante n'est appropriée, il est possible d'en créer une nouvelle de façon dynamique.

3.6 Les techniques de migration

Lors de l'application des changements sur le schéma d'un processus métier, se pose naturellement le problème de la gestion des instances de ce processus qui sont en cours d'exécution au moment du changement. Dans cette perspective, les approches de migration sont des techniques d'analyse d'impact du changement qui visent à définir, et à analyser les effets sur les instances en cours d'exécution quand des changements sont effectués sur la définition d'un processus métier. Parmi ces approches, nous distinguons l'approche par annulation, l'approche sans propagation des changements et l'approche avec propagation des changements.

3.6.1 Approche par annulation d'exécution

Dans le cadre de cette approche radicale, les instances en cours d'exécution qui sont affectées par les changements du processus métier sont simplement annulées et redémarrées dés le début. Cette façon de faire permet de créer de nouvelles instances qui sont en conformité avec la nouvelle spécification du processus. Malgré sa simplicité, cette technique est la moins recommandée en raison de la perte du temps et des ressources déployées (*occupations des utilisateurs, matériels, perte d'informations*). En effet, elle exige une reprise complète des activités qui ont été déjà accomplies, d'où une perte considérable de travail.

3.6.2 Approche sans propagation des effets

Cette approche est basée sur la co-existence de l'ancienne et de la nouvelle version du processus métier. Ayant les deux versions du processus métiers qui sont actives simultanément, les instances en cours d'exécution continueront leurs exécutions sur

la base de l'ancienne définition du processus métier, alors que les nouvelles instances démarreront leurs exécutions conformément à la nouvelle version. Le gestionnaire du processus métiers pourra par la suite choisir le moment opportun pour opérer la propagation des effets des changements sur les instances en cours d'une manière différée. Par exemple lorsque les instances achèvent complètement leurs exécutions ou lorsque le système est off-line.

Un exemple de telles approches est cité dans le travail de Weske M. [115]. L'approche est basée sur un méta-modèle métier, pour lequel l'auteur propose une modélisation de l'exécution flexible des différentes activités d'un workflow. Plusieurs type de changement, tels que l'ajout ou la suppression d'activités, sont gérés alors que des instances sont en cours d'exécution. Cependant, la technique exige que l'activité à modifier ne doit pas être en état d'activation au moment de l'application du changement. Cette restriction forte peut entraver la bonne mise en œuvre de l'approche. La même approche a été également utilisée dans l'outil Milano proposé dans le travail de Agostini et al [116], ou encore l'outil WASA2 proposé dans les travaux de Weske M. [117].

En se basant sur le même principe, dans [13] les auteurs proposent un mécanisme de gestion de versions pour les différents schémas du processus métier, en représentant les différents dépendances fonctionnels entre schémas.

3.6.3 Approche avec propagation des effets

Afin de refléter d'une manière fidèle les changements opérés sur les processus métiers, les effets engendrés doivent être propagés de manière naturelle et transparente aux instances en cours d'exécution. Pour atteindre ces objectifs, des techniques plus élaborées doivent être déployées. Ces techniques nécessitent un modèle de transition permettant la mise conformité des instances actuelles, ayant démarré sur la base d'un ancien modèle de processus, avec la nouvelle définition du processus évolué [23].

Plusieurs travaux de recherche ont abordé la migration des instances en cours d'exécution avec le principe de la propagation des effets des changements. A titre illustratif, Casati et al présentent dans [16] un langage de gestion du changement des workflows ; appelé "WFML". Le langage répond à la question de la migration des instances en cours d'exécution quand la définition du processus métier évolue, et l'approche proposée exploite un ensemble de critères formels pour déterminer l'ensemble des instances qui peuvent migrer en toute sécurité vers le nouveau schéma de processus métier.

L'outil $ADEPT_{FLEX}$ présenté dans [118], permet l'intégration des changements d'une façon dynamique et fluide, même durant l'exécution des instances et sans perdre ni le contrôle ni la cohérence structurelle de ces instances. Les auteurs proposent une politique migratoire qui prend en charge la gestion des conflits potentiels qui peuvent être causés par les opérations de changements.

3.7 Critères retenus pour les processus métiers

Avant de terminer ce chapitre, nous énonçons quelques critères retenus dans le cadre de nos travaux et qui sont associés aux processus métiers. Ces critères constituent des repères qui vont baliser la poursuite de la thèse.

- **Implémentation des processus métiers :** du point de vue fonctionnel, un processus métier exprime un savoir faire qui peut être effectué manuellement ou géré automatiquement par un **BPMS**, ou un **WfMS**. Dans cette thèse nous allons nous intéressés, particulièrement, aux processus métiers implémentés en tant que services web.
- **Les opérations de changement :** vue la diversité des opérations de changement, par la suite nous ne faisons aucune restriction sur leur type (*basiques vs complexes*), ni sur leur portée (*schéma vs instances*).
- **Causes et effets des changements :** Nous nous intéresserons seulement aux effets des changements sans tenir compte de leurs causes. Par la suite, nous considérerons seulement les changements et leurs effets sur les processus métiers, abstraction faite aux raisons qui ont induit ces changements.
- **Flexibilité vs migration :** Contrairement aux approches de flexibilité, qui supposent une connaissance préalable des différentes possibilités d'exécution des processus métiers, la migration permet de faire basculer les instances en cours dans le nouveau schéma du processus.
  A noter que dans nos travaux, nous allons nous intéressés à la migration en prenant en compte la propagation des effets engendrés par l'application des changements, tels que décrits dans la section (cf., 3.6.3).

3.8 Conclusion

Afin d'atteindre leurs objectifs, et assurer leur pérennité dans un contexte extrêmement concurrentiel, les entreprises modernes ont de plus en plus besoin de gérer des processus métiers complexes qui peuvent s'étendre au delà des frontières de l'organisation. Cette exigence passe, inévitablement, par une prise en main rigoureuse de la dimension processus métiers et leur adaptation aux perpétuels changements.

Dans ce chapitre nous avons introduis les concepts liés aux processus métiers, et nous avons mis en exergue les besoins de gérer leur cycle de vie, particulièrement, la nécessité de prendre en charge leur évolution. Par ailleurs, la jonction entre les processus métiers et les services web a été mise en évidence.

Comme les entreprises modernes gèrent des processus métiers complexes qui évoluent dans des environnements fortement versatiles, nous avons présenté les notions inhérentes aux changements de processus métiers, tout en exposant la typologie des opérations de changement associées et leurs propriétés. Une attention particulière a été accordée à l'impact du changement des processus métiers, et nous avons exposé quelques techniques d'analyse de l'impact du changement.

Enfin, nous avons préciser les critères retenus dans le cadre de cette thèse.

Dans le prochain chapitre nous allons explorer, d'une manière détaillée, les travaux ayant traité la problématique des évolutions et de la gestion de l'impact des changements.

Chapitre 4

Travaux connexes

4.1 Introduction

Le problème de l'évolution des applications informatiques au sens large a été étudié par différents domaines de la discipline informatique. A cet égard, et en raison de sa nature multidisciplinaire, la question de l'évolution des processus métiers et des services web est à la croisée de chemin de plusieurs domaines de recherche. Force est de constater que la littérature existante est très riche en travaux qui ont tenté de relever les défis inhérents à cette problématique de recherche.

Ce chapitre est dédié à l'exploration, à l'étude et à l'analyse des travaux connexes ayant traité, d'une manière ou d'une autre, les questions des changements et des évolutions.

Nous commençons le chapitre par la section 4.2 qui expose une analyse détaillée des différents approches générales ayant traité les problèmes des évolutions et leurs impacts sur les schémas et/ou sur les instances en cours d'exécution. Ensuite, une étude comparative détaillée des divers travaux et leurs relations avec notre problème est présentée. Enfin, les limites des approches générales existantes sont exposées et les raisons de leur non adéquation au contexte de notre problème sont justifiées.

Dans la section 4.3, l'accent est mis particulièrement sur l'analyse des approches existantes pour la gestion de l'évolution et de la migration dans le cadre des protocoles des services web et leurs insuffisances sont mises en évidence.

Enfin, une exploration de quelques outils logiciels existants dans le domaine industriel pour la gestion de la migration est présentée dans la section 4.4. Nous terminons le chapitre par une conclusion, donnée en section 4.5, qui fixe les repères et les lignes directrices pour la deuxième partie de la thèse, en termes d'exigences et de critères à satisfaire lors de la spécification des stratégies de gestion de la migration des instances actives des processus métiers des services web.

4.2 Les approches générales pour la gestion des évolutions

Des approches très variées ont été proposées dans divers domaines de la discipline informatique pour répondre aux questions liées aux évolutions, à la gestion de l'impact des changements et à la migration d'instances. Cependant, ces approches restent de manière naturelle très liées au domaine d'étude en question (*données, logiciels, processus, ..., etc*). Autrement dit, elles se focalisent, généralement, sur des aspects particuliers du problème de l'évolution (*migration de données, suivi des versions, résolution de conflits, génération d'adaptateurs, ..., etc*).

A titre d'exemple, dans le domaine connexe de la gestion des workflows [1] et des processus métiers, la littérature consacrée aux problèmes de la gestion des changements est extrêmement riche. Elle soulève plusieurs difficultés de natures différentes et dont la plupart ont été largement étudiés. Ainsi, différentes facettes de l'évolution des schémas des processus métiers ont été explorées, comme par exemples : la propagation des changements [12], le contrôle des versions d'un processus[13], les patrons d'adaptation [14] ou encore les primitives de modification et leurs propriétés [15].

Toujours dans le même domaine, la gestion des changements dynamiques des schémas de workflows a été profondément traitée dans [18], et une approche pour calculer les régions du schéma d'un workflow qui sont affectées par les changements a été proposée. Dans cette approche, seules les instances actives qui se trouvent en dehors des régions affectées par le changement sont classées comme instances *migrables*. Dans [19], un algorithme de calcul de la zone de changement est proposé, et dans les travaux de [17] un modèle générique pour toute une famille du même processus workflow est décrit et modélisé.

4.2.1 Etude des travaux des domaines connexes

L'étude de l'état de l'art des travaux connexes fait ressortir toute une variété de domaines qui ont traité, d'une manière ou d'une autre, le problème de l'évolution sous différents angles. Nous avons identifié, notamment, les sept domaines suivants :

- Évolution des schémas des bases de données : [119, 120, 121]
- Évolution des ontologies :[122, 123, 124, 125] ;
- Evolution des composants logiciels :[126, 127, 128]
- Domaine de la ré-ingénierie des logiciels :[129, 130, 131]
- Domaine de la migration des systèmes patrimoniaux : [132, 133]
- Évolution des workflows : [12, 13, 14, 15, 16, 17, 18, 19, 20]
- Évolution des protocoles : [10, 11, 24, 25, 134, 135, 136, 101]

Dans ce que suit, nous explorons les différents domaines connexes qui ont traité le problème de l'évolution, et nous discutons en détails les limites des différentes approches proposées dans chaque domaine.

a) Domaines des bases de données et des ontologies

La communauté des bases de données et celle des ontologies ont considéré, principalement, le problème de contrôle des évolutions par rapport à la variation des schémas des structures afférentes. Ainsi, pour répondre aux nouvelles exigences des applications, la définition du schéma de la base de donnée ou de l'ontologie est modifiée avec le temps, en ajoutant ou en enlevant des éléments du schéma. Cependant, les approches proposées dans le cadre de la gestion de l'évolution des schémas de base de données [119, 120, 121] et celles abordant les évolutions des ontologies [122, 123, 124, 125] maintiennent, simultanément, anciennes et nouvelles versions de leurs schémas opérationnelles, même après les modifications des spécifications. Le principe de la gestion de l'évolution est fondé sur des techniques de mapping, telles que celles proposées dans

1. Le terme workflow est utilisé pour désigner des "flux de travaux", ou "flux opérationnel".

[137, 138], qui seront déployées plus tard pour effectuer la transformation des données de l'ancien schéma vers le nouveau.

Dans ce type d'approches, il est observé qu'anciens et nouveaux schémas des bases de données ou des conceptualisations des ontologies cohabitent ensemble durant le processus de mapping. C'est à cause de cette raison que ces approches demeurent limitées pour gérer les évolutions dynamiques dans le contexte des protocoles des services web. En effet, ces dernières évolutions ne tolèrent l'existence que d'un unique schéma (*modèle*) qui reflète la nouvelle version du protocole de service. La deuxième raison est que dans le cadre des processus métiers des services web, les descriptions y afférentes pourraient manifester des comportements, où il serait impossible pour toutes les instances actives de poursuivre leur exécution en raison des changements opérés. En particulier, c'est le cas lorsque de nouvelles exigences sont incontournables, et doivent être respectées par certaines instances ciblées. Cette situation engendre un besoin d'une migration sélective qui est fondée sur des configurations concrètes. Ces dernières sont exprimées par les traces des exécutions réelles des instances actives. Or, dans le contexte des bases de données, les processus de migration sont globaux, et affectent toutes les instances en cours sans exceptions. Une autre insuffisance de ces deux domaines concerne le choix du moment du changement. En effet, l'administrateur de la base de données ou de l'ontologie a toujours la possibilité de choisir l'instant propice pour opérer les changements. Par contre, cet avantage n'est pas permis pour les protocoles des processus métiers qui exigent une entrée immédiate des modifications associées à certaines règles métier.

De ce qui précède, nous concluons que les mécanismes proposés dans le cadre des évolutions des bases de données et des ontologies ne peuvent pas être exploités dans notre contexte d'évolution dynamique des protocoles des services web.

b) Domaine de l'évolution des composants et ré-ingénierie des logiciels

En partant de l'hypothèse qu'un service web est une application informatique, les techniques de gestion des évolutions proposées dans les domaines de la ré-ingénierie des logiciels (*software re-engineering*)[129, 130, 131] et celui de l'évolution des composants logiciels [126, 127, 128] ont été étudiées pour envisager leur adaptation à notre contexte d'évolution des protocoles des services web.

L'exploration approfondie de ces deux domaines a fait ressortir que de ces approches tolèrent l'existence simultanée, dans un système en plein évolution, de plusieurs versions de la même application ou du même composant logiciel. En ce sens, et afin d'associer les informations spécifiques des différentes versions, les techniques de versioning s'appuient sur l'amélioration des noms de fichiers des bibliothèques par les numéros de version, aussi bien que sur l'enrichissement des noms des bibliothèques par des méta-fichiers spéciaux (*i.e., les fichiers manifestes d'un format XML*). Ainsi, de tels mécanismes permettent à des versions multiples d'un composant d'exister dans un système, et donnent la possibilité aux applications d'employer les versions les plus adéquates. Dans un tel contexte, les utilisateurs auront toujours l'avantage d'utiliser différentes versions simultanément. Néanmoins, les protocoles métiers des services web mettent l'accent uniquement sur la dernière version du protocole de service afin de satisfaire les nouvelles exigences exprimées par de nouvelles règles métiers. En

plus, de part la nature du domaine lui même (*programmation, tests, intégration*), les techniques qui sont proposées sont statiques, et se concentrent essentiellement sur les aspects structurels et fonctionnels des programmes.

Il en résulte que les approches proposées par les deux domaines précédents restent limitées pour surmonter les difficultés rencontrées lors de la gestion de l'évolution des protocoles des services web, et elle ne peuvent pas gérer de manière conséquente la migration des instances actives.

c) Domaine de la migration des systèmes patrimoniaux

Par analogie à la migration des systèmes patrimoniaux (*legacy systems*), l'évolution d'un service web peut être vue comme une application obsolète qui doit subir des mutations afin de s'adapter aux nouvelles exigences technologiques et environnementales. Cependant, l'analyse du domaine de la migration des systèmes patrimoniaux [132, 133] fait ressortir le fait que les techniques de transformation qui sont proposées sont très complexes et très couteuses. Par ailleurs, elles sont spécifiques à des systèmes d'informations particuliers. Du point de vue gestion des instances d'exécution, celles ci ne sont pas gérées dans de tels systèmes. De plus, les approches existantes sont très variées et elles sont, souvent, très divergentes d'un système à l'autre.

Généralement, la migration des systèmes patrimoniaux exige des adaptateurs adéquats qu'il faut déployer au cas par cas. Or, dans notre contexte nous visons une approche globale qui doit prendre en charge la migration dynamique des instances actives dans un cadre commun basé sur des **standards reconnus** qui est celui des services web.

d) Domaine des workflows

Comme les services web sont l'aboutissement naturel des workflows traditionnels, nous pouvons prétendre utiliser les techniques proposées par ce domaine pour la gestion de l'évolution, et les exploiter dans le cadre de la gestion de l'évolution des protocoles des services web. Effectivement, la littérature existante dans ce domaine est très variée et très riche, et plusieurs défis ont été relevés, y compris celui de l'analyse de l'impact et de la propagation des changements. Dans ce qui suit, nous passons en revue les travaux les plus pertinents.

Plusieurs travaux ont traité divers aspects du problème de l'évolution des processus métiers ou des workflows [12, 13, 14, 15, 23, 22]. D'autres travaux se sont intéressés aux questions relatives à l'évolution des schémas des processus métiers/workflow, e.g., contrôle de versions des processus [13], spécification des primitives de modification et les propriétés associées [15], proposition des patrons d'adaptation [14] et prise en compte de la propagation des changements [12]. Cependant, une grande majorité des approches proposées s'est concentrée, essentiellement, sur la vérification de critères de conformité afin d'assurer que la migration d'instances ne souffre pas de problèmes, tels que l'inter-blocage ou la suppression d'une activité déjà exécutée (*voir par exemple [23] pour un état de l'art plus détaillé*).

Une autre catégorie des travaux relatifs aux changements dynamiques des workflows s'est focalisée sur la problématique de la migration proprement dite [16, 17, 18, 19, 20, 139]. L'analyse de ces travaux fait ressortir le fait que toutes les techniques

proposées se basent sur le principe du **report de la migration**, tout en maintenant pour des raisons opérationnelles, l'ancienne et la nouvelle version du workflow fonctionnelles en même temps. Par ailleurs, certains travaux restreignent leurs analyses à la seule région affectée par les changements, tout en la calculant d'une manière manuelle [18, 19]. D'autres travaux proposent des algorithmes de calcul des régions de changement, et décrivent des modèles de processus métiers qui sont génériques pour toute une famille de variantes d'un même processus [17].

Une approche remarquable est exposée dans [139], où un langage de définition des politiques de migration est proposé pour traiter les changements des processus métiers. Ces politiques permettent de définir des règles de re-configuration qui spécifient les changements structurels de courte durée pour une instance d'un workflow. En fin, des travaux plus conséquents proposent de créer une nouvelle version du workflow à laquelle seront rattachée les instances actives qui pourront continuer leur exécution conformément à cette nouvelle version [15]. Néanmoins, cette approche devient encombrante si les changements sont fréquents et potentiellement prévisibles. Le constat général est que dans ce type d'approches inhérentes aux workflows, les instances actives sont gérées d'une manière simple en noir et blanc (*migrables, non migrables*), et les besoins spécifiques de migration pour des instances particulières ne sont pas pris en compte.

En conclusion, les techniques de gestion des évolutions proposées dans le cadre des workflows ne peuvent pas propager, d'une manière conséquente et efficace les changements, et par conséquent, elles sont spécifiques et limitées. En définitif, les approches utilisées par les workflows ne peuvent être exploitées pour gérer la migration des instances actives d'un service web qui a subi des évolutions.

e) Domaine de l'évolution des protocoles

Dans le domaine des évolutions des protocoles, tels que les protocoles de communication et les protocoles de sécurité, différentes approches ont été développées. Ces approches vont de la translation basée sur les évènements [134] aux protocoles de négociation de confiance [135]. Cependant, les approches proposées s'intéressent, de part leur nature, au niveau de communication le plus bas [1]. Contrairement au contexte des évolutions des protocoles des services web, nous nous intéressons aux protocoles métiers qui se situent au niveau le plus élevé de la pile conceptuelle d'interopérabilité (*voir figure 2.1*). Ainsi, le niveau de préoccupation est complètement différent, et notre objectif est de résoudre les problèmes d'interactivité et de conversations, au lieu de traiter des questions liées au transport de messages et à la communication.

A noter que le point fort des approches relatives aux évolutions des protocoles concerne la prise en charge des instances compatibles, par la mise en œuvre d'une chaine d'adaptateurs spécifiques [135]. Néanmoins, cette solution devient, rapidement, encombrante si les changements sont fréquents car une chaine complexe d'adaptateurs est requise. La gestion de cette chaine devient, alors, très couteuse et très lourde.

Bien que les approches proposées dans [135, 140] ont tenté de surmonter les difficultés afférentes aux négociations de confiance, les contraintes à gérer dans notre problème soulèvent des difficultés particulières aux protocoles des processus métiers, et rendent l'analyse de l'impact plus complexe.

4.2.2 Analyse comparative des approches générales existantes

Le **Tableau. 4.1** expose une étude comparative des travaux de recherche ayant abordé, plus ou moins, les questions de la gestion des changements et des évolutions.

Lors de cette étude comparative, les critères suivants ont été retenus afin de faire ressortir les points faibles et les points forts de chaque approche.

- **Les modèles utilisés** : ce critère décrit les modèles formels et descriptifs manipulés par chaque approche pour gérer les changements.
- **Les opérations de changement** : désigne les préoccupations et les objectifs visés par les changements.
- **Nature de l'évolution** : statique ou dynamique.
- **Les éléments traités** : spécifie le composant sur lequel agissent les changements.
- **Gestion des instances** : explique si le domaine en question gère les instances ou non, et comment sont-elles prises en charge à la suite des opérations de changement.
- **Processus de migration** : illustre les techniques utilisées pour gérer les instances actives (*manuelle, automatique, différée, ...*)
- **Limites constatées** : Cette colonne met en exergue les insuffisances et les points faibles qui entravent l'exploitation de l'approche dans notre contexte.

A partir de l'étude précédente, il apparait clairement que chacun des domaines analysés présente plusieurs points communs avec notre problème. Néanmoins, les approches qui sont proposées présentent plusieurs insuffisances pour la prise en charge des problèmes liés à la gestion des évolutions dynamiques des protocoles des services web (*voir dernière colonne*).

En résumé, les approches proposées dans les différents domaines connexes ne peuvent être appliquées dans le cadre de la gestion des évolutions dynamiques des protocoles des services web pour les principales raisons suivantes :

- Dans le contexte de l'évolution dynamique des services web, plusieurs instances sont en cours d'exécution au moment du changement.
- Seule la dernière version du protocole doit être opérationnelle. Les anciennes versions sont considérées comme obsolètes, car elles ne reflètent plus les processus métiers évolués.
- Le processus de migration doit être instantané et ne peut être différé. Autrement dit, les nouvelles règles de gestion doivent entrer instantanément en application.
- Du fait que des ressources critiques sont déployées lors de la migration, cette dernière doit être sélective, et toutes les instances ne peuvent pas migrer vers la nouvelle version du protocole.

Critères → Domaines ↓	Modèles utilisés	Opérations de changement	Nature de l'évolution	Éléments traités	Gestion d'instances	Processus de migration	Limites constatées
Evolution des BDD, versions de schémas	-Relationnel -objet	Ajout/ suppression d'éléments	Dynamique	Schéma, et données	Les applications peuvent continuer leurs exécutions	Transfor- mation, vues, mapping	-Migration globale -anciennes versions actives
Evolution d'ontologies	Logiques de description (LD)	Mise à jour des relations, des classes et méta-data	Statique/ dynamique, inverser les effets	Concepts, instances, données, axiomes	-Re-classification -notification -résolution de conflits	-Alignement -similarités -propagation des changements	-Maintient toutes les versions, gère les contraintes, effets inverses
Evolution de composants logiciels	Prototype, itérative, spirale, agile, incrémentale	Mise à jour classes, méthodes, variables	Statique/ dynamique	Activités, données, règles métiers	Les applications sont rattachées aux versions	Versioning, connecteurs, liens dynamiques (DLL)	Anciennes versions utilisées -perte de dynamicité
Ré-ingénirie de logiciels	-Diagrammes UML -graphes	Changement de la structure interne des programmes	Statique (modularité, réutilisation)	Modules, structures	Après déploiement des nouvelles applications	Non concerné -fonctions d'édition -restructuration	-Gère seulement, les changements structurels et statiques
Workflows	Réseaux de Petri AFD	Modification des schémas	Dynamique, à la volée	Flux d'activités	Région modifiée, classification, versions	-Différée -adaptateurs semi- automatique	-Migration en différée - les versions anciennes sont actives
Evolution des protocoles	-Réseaux de Petri, AFD -diagrammes- UML	Améliorer le schéma du protocole, les politiques	Statique / dynamique	Activités, politiques	-Instances compatibles seules, -chaines d'adaptateurs	-Adaptation -substitution -versions -namespace	Non efficace pour les évolutions fréquentes, ada- ptateurs complexes
Migration des systèmes patrimoniaux	Divers modèles (données, logiciels)	Application de nouvelles règles métiers, adaptation	Statique	-Logiciels -Base de données	-Transfor- mation, -adaptation	-Semi-automa- tique - cas par cas, -mapping,wrapper -substitution	-Complexe -couteuse -les instances non gérées

TABLE 4.1 – Etude comparative des approches générales pour la gestion des évolutions

4.3 Les travaux spécifiques à l'évolution des protocoles des services web

Après avoir explorer les divers domaines ayant traiter d'une manière générale la question des évolutions et des changements, dans cette section nous limitons notre étude à l'analyse des travaux ayant, étroitement, abordé le problème des changements et leurs impacts. Par la suite, nous restreindrons notre étude au contexte spécifique des évolutions des protocoles des services web.

4.3.1 Étude des approches pour la gestion de l'évolution des protocoles de services

L'analyse des protocoles des services web et leurs évolutions a profité de beaucoup de contributions qui varient de la vérification des schémas des protocoles [10, 11, 134, 141], à la substitution des protocoles [64, 136], ou encore les techniques de gestion des adaptateurs [135]. Nous pouvons classer les travaux existants en deux grandes catégories : les travaux basés sur l'analyse de compatibilité et de remplaçabilité, et les travaux basés sur les techniques d'adaptation.

Dans ce qui suit, nous examinons ces travaux d'une manière approfondie.

a) Travaux basés sur l'analyse de compatibilité et de remplaçabilité

Ce type de travaux se concentre, essentiellement, sur le principe de la préservation de la compatibilité des schémas des protocoles des services web, après l'évolution de leurs spécifications. Dans le cas où des protocoles ayant subi des changements deviennent incompatibles, des techniques de substitution sont proposées pour rechercher des protocoles de remplacement.

• Pour gérer le cycle de vie des services web, les auteurs dans [10, 11] présentent un cadre général adéquat à la modélisation et à l'analyse de la compatibilité des protocoles de services. Des mécanismes de substitution des protocoles sont proposés pour remplacer ceux qui ont évolué et/ou qui sont devenus incompatibles avec les nouvelles spécifications, et qui ne pourront plus supporter les mêmes conversations que celles exposées par les protocoles initiaux. Dans [61], la description des protocoles est enrichie par des contraintes temporelles et le même type d'analyse est réalisé. Les contraintes transactionnelles sont formalisées et différentes analyses de compatibilité sont étudiées dans [62]. Néanmoins, toutes les analyses précédentes sont limitées au **contexte statique**, et ne prennent pas en compte les besoins particuliers des instances qui sont actives au moment des changements.

• Dans [14], les auteurs proposent un ensemble d'opérateurs de changement de base qui peuvent être appliqués aux protocoles, et différents patterns d'évolution sont explicités. Néanmoins, les changements sont abordés au niveau du schéma des protocoles, et aucune analyse de l'impact du changement sur les instances en cours d'exécution n'a été réalisée.

• Certaines approches existantes pour la gestion de l'**évolution dynamique** des protocoles des services web [21, 24, 25, 97, 136, 142] utilisent un modèle de protocole de processus métier qui est basé sur des automates d'états finis déterministes. En se basant sur ce modèle, les techniques proposées étudient d'une manière consistante le problème de la migration des instances lors de l'évolution des schémas des protocoles. Un objectif principal poursuivi par ces approches réside dans l'identification des instances qui ne sont pas affectées par les changements. En conséquence, ces instances peuvent migrer sans difficultés vers le nouveau protocole, et le processus de migration reste transparent pour les clients du service.

L'idée de base dans ce type d'approches consiste à utiliser les notions de compatibilité et de remplaçabilité proposées dans [10, 11], comme principe pour identifier les instances qui peuvent être transférées en toute sécurité, lorsque le schéma du protocole a évolué. Deux principales notions de compatibilité ont été définies afin de soutenir l'analyse fine des impacts de l'évolution d'un protocole : *la compatibilité historique* et *la compatibilité future*.

La compatibilité historique permet d'assurer que *l'historique* d'une instance i est compatible avec le nouveau protocole (*i.e., les exécutions passées de i peuvent être rejouées dans la nouvelle version du protocole*). Par contre, *la compatibilité future* traite des exécutions futures d'une instance i, et permet de veiller à ce que l'ensemble des chemins d'exécution futures *(les chemins qui commencent à partir de l'état actuel de l'instance i)*, soit inclus dans l'ensemble des chemins d'exécution futures commençant à partir de l'état correspondant dans le nouveau protocole.

Dans [24, 25], une analyse basée sur le modèle de protocole précédent est conduite. Elle prend la forme d'une hiérarchie de classes de remplaçabilité pour gérer la migration des instances du protocole évolué. La classe la plus contraignante ; *remplaçabilité complète* implique une forte relation entre un protocole $\mathcal{P}$ et sa nouvelle version $\mathcal{P}'$, ce qui nécessite que l'ensemble des chemins d'exécution de $\mathcal{P}$ soit inclus dans l'ensemble des chemins d'exécution de $\mathcal{P}'$. Par conséquent, dans ce cas de *remplaçabilité complète*, toutes les instances de $\mathcal{P}$ sont, à la fois, compatibles dans le *passé* et compatibles dans le *futur* avec $\mathcal{P}'$. Donc, elles peuvent être migrées en toute sécurité vers $\mathcal{P}'$.

Dans le cas où *la compatibilité complète* n'est pas satisfaite entre $\mathcal{P}$ et $\mathcal{P}'$, d'autres classes de remplaçabilité moins contraignantes peuvent être utilisées. Par exemple, *la remplaçabilité par rapport à un état* vise à calculer les états de l'ancien protocole qui ne sont pas affectés par les changements. Un état s de $\mathcal{P}$ est dit non affecté par les changements, s'il possède les mêmes chemins passés et les mêmes chemins futurs que son état s' correspondant dans la nouvelle version $\mathcal{P}'$. Les chemins passés d'un état s sont tous les chemins qui partent de l'état initial de $\mathcal{P}$ jusqu'à l'état s et les chemins futurs d'un état s sont ceux qui commencent à partir de s, et se terminent à un état final de $\mathcal{P}$. Par conséquent, les instances en cours qui sont dans des états non affectés par les changements peuvent être migrées d'une manière transparente vers le nouveau protocole. En plus, la notion de remplaçabilité est raffinée en vue de traiter *la remplaçabilité des chemins*, où deux sous-classes sont distinguées : *la remplaçabilité des chemins passés* qui comprend les états de $\mathcal{P}$ qui ont un ensemble de chemins d'exécution passés inclus dans l'ensemble des chemins d'exécution passés d'un état s' correspondant dans $\mathcal{P}'$, et *la remplaçabilité de chemins futurs* qui comprend les états de $\mathcal{P}$ qui ont un ensemble de chemins d'exécution futurs inclus dans l'ensemble des

chemins d'exécution futurs d'un état correspondant s' de $\mathcal{P}'$.

L'insuffisance principale des approches précédentes réside dans l'aspect gestion de la compatibilité future. En effet, celle ci est basée sur les techniques de fouilles de données pour l'inférence logique du schéma du sous-protocole exprimant des interactions futures. Ce schéma est déduit par l'exploitation de la base des traces d'exécution stockée dans les journaux logs. Cependant, il est possible pour certaines instances actives d'emprunter, dans le nouveau protocole, des chemins d'exécution qui n'ont jamais été exécutés par des instances déjà actives ou terminées (*aucune trace historique*). En effet, des conversations peuvent être engagées pour la première fois, sans qu'elles n'aient des traces similaires dans les fichiers logs. Cette situation rend la déduction du sous-protocole futur imprévisible. En plus, ces approches sont purement algorithmiques, et ne prennent pas en compte les besoins spécifiques des instances en cours. Cette façon d'aborder le problème limite, considérablement, la dimension de la compatibilité au sens le plus **strict**.

[21] étend les techniques de [24] au cas des protocoles de service asynchrones et non-déterministes.

b) Travaux basés sur les techniques d'adaptation

Dans ce type travaux, l'accent est mis sur le calcul et la prise en charge des différences entre les protocoles et/ou les interfaces résultant de la mise en œuvre des changements.

• Dans le contexte de l'adaptation des protocoles métiers des services web, certains travaux [141, 143] proposent des techniques pour calculer les différences entre les protocoles pour les exploiter lors de la génération des adaptateurs qui comblent les écarts entre les versions de protocoles. Cependant, ce type de mécanismes devient très lourd si les changements sont fréquents, et engendre, par là même, un nombre important d'adaptateurs. Cette situation exige des efforts considérables pour gérer la connexion de la chaine des adaptateurs qui sont requis. En plus, l'activation des adaptateurs est souvent réalisée lorsque le système est à l'arrêt *(off-line)*.

• Dans [97], l'auteur introduit deux types de changement, à savoir, *les changements superficiels*, pour lesquels les modifications se limitent aux interfaces de services, et *les changements profonds*, pour lesquels des effets en cascade et des effets secondaires sont produits. Ce travail présente un cadre théorique, basé sur les notions de compatibilité des versions, pour faire face aux changements superficiels de services. Le cadre est basé sur un accord (*agreement*), sous forme de contrat partagé, entre le fournisseur de service et le client. Nous estimons que cette approche est trop encombrante à cause des contrats itératifs qu'il faut élaborer. Ainsi, elle ne répond pas d'une manière adéquate et consistante au problème des évolutions, du fait que les contrats ne peuvent être négociés d'une manière répétitive au moment de l'exécution, surtout lorsque le nombre d'instances du service est très élevé. Par contre, une méthodologie de conception orientée changements est proposée pour gérer le cycle de vie d'un service, tout en prenant en compte les effets des changements profonds qui peuvent affecter les services.

• Dans [101], l'auteur propose un ensemble de contributions permettant une vérification de la cohérence et de la conformité des modèles de processus métier après chaque changement, mais aussi d'établir une évaluation *a priori* de l'impact structurel

et qualitatif des modifications. L'analyse de l'impact est basée sur la relation générique dite de dépendance. En effet, une telle relation permet de dire : *si une entité **B** dépend d'une autre entité **A** alors le changement de A aura un impact sur B*. Néanmoins, dans ce travail l'analyse de l'impact est abordée au niveau schéma du processus métier sans tenir compte des effets sur les instances qui sont en cours d'exécution.

• En se basant sur les travaux de [24, 25], les auteurs dans [144] abordent les problèmes des évolutions en comparant les interfaces de services, mais sans tenir compte des spécifications relatives aux protocoles de services.
- Dans le même contexte, dans [145] les auteurs proposent une technique pour l'analyse des différentes versions successives d'une interface de service, dans le but de détecter les changements et les incohérences. Néanmoins, le travail en question se limite à déceler les incompatibilités des versions, tout en produisant un diagnostique des détails.
- Dans [146], les auteurs proposent une méthode pour déterminer automatiquement quand deux descriptions de services sont compatibles. Ils évaluent, par la suite, l'impact des changements sur l'interface. Cette approche se base sur l'interface de service, et ne prend en compte que le cas de la compatibilité passée.
- Dans [147], les auteurs présentent un cadre général pour la découverte des profils des utilisateurs d'un service. Par la suite, ce profil sera utilisé pour caractériser les groupes d'utilisateurs et leurs besoins spécifiques en fonctionnalités du service. Cette méthode reste limitée, car l'impact des changements est considéré par rapport aux incompatibilités des profils des seuls clients qui sont déjà enregistrés dans le système. Cela exige une mise à jour du système à chaque manifestation d'un nouveau client.
- Dans [148], les auteurs se concentrent sur l'analyse des dépendances entre les services et les processus métiers qui les supportent. Ils présentent une classification des changements qui peuvent affecter les processus métiers. Un ensemble de patterns de changement est identifié sur la base des types de dépendances et une analyse de l'impact des changements est proposée pour gérer leur propagation sur les services associés.
- Dans [149], l'analyse de l'impact est conduite sous divers angles. L'analyse des dépendances, l'analyse de la traçabilité et l'analyse des historiques. Les auteurs concluent que la majorité des outils existants pour d'analyse de la propagation des changements est encore semi-automatique.

4.3.2 Synthèse des travaux sur les protocoles des services web

Les approches proposées dans le cadre de l'évolution des protocoles de services web ont abordé la question de l'analyse de l'impact des changements dans différentes perspectives : niveau protocole du service, par rapport aux interfaces de services et du point de vue gestion des instances en cours d'exécution.

Malgré les contributions majeurs apportées par les approches existantes pour la gestion de l'impact du changement des services web, elles se basent toutes sur des **stratégies de migration prédéfinies**. La conséquence immédiate est que les gestionnaires de protocoles ne peuvent pas définir leur propres stratégies de migration qui répondront à des besoins spécifiques de migration .

Nous récapitulons, dans le **Tableau. 4.2**, les insuffisances majeures des approches précédentes. La synthèse est basée sur des caractéristiques pertinentes que nous avons identifié durant notre étude des travaux existants (*première colonne*).

Caractéristiques	Insuffisances constatées
les types de compatibilité traités	compatibilité passée, future, complète
les types de changements	changement basiques (*états, messages, interfaces*)
les aspects pris en charge	contraintes d'obligation uniquement
type de stratégie de migration	prédéfinie (*non paramétrable*)
spécification de la compatibilité	± formelle compatibilité au sens strict (*non flexible*)
fondement des approches	algorithmique ± formel

TABLE 4.2 – Synthèse des travaux sur les évolutions des protocoles des services web

Les insuffisances constatées, telles qu'illustrées dans la deuxième colonne du **Tableau. 4.2**, constituent des lignes directrices pour la proposition d'une nouvelle approche qui prendra en charge, de manière effective, les différentes questions soulevées lors de la gestion des évolutions dynamiques des protocoles des services web.

Pour surmonter les limites constatées, la feuille de route qui guidera notre contribution doit porter sur une approche de gestion des changements qui sont de nature complexe. L'approche à proposer doit, également, offrir un cadre formel qui soutiendra la spécification des stratégies de migration plus flexibles. Par ailleurs, l'approche doit être, à la fois, générique, paramétrable et capable de gérer différents types de compatibilité (*totale et/ou partielle, passée et/ou future*).

4.4 Exploration des outils du domaine industriel

Dans le monde industriel, la question de la migration des instances actives, connue également sous le nom de *la gestion des instances à la volée (in-flight instances management)*, est reconnue comme l'un des problèmes les plus difficiles auxquels la technologie BPM a été confrontée [2].

Dans cette section nous présentons, d'une manière non exhaustive, quelques environnements logiciels de gestion des processus métiers (**BPM**), tout en mettant l'accent sur l'aspect gestion de la migration des instances actives des processus métiers.

4.4.1 Étude de quelques outils logiciels existants

Plusieurs fournisseurs de logiciels proposent différents outils pour la gestion du cycle de vie des processus métiers. Ces outils constituent des environnements de travail complets qui offrent plusieurs fonctionnalités intégrant la description des processus métiers, la gestion de l'évolution, le stockage des traces et même la supervision des exécutions (*monitoring*).

Les outils les plus populaires rencontrés sont :

1. IBM webSphere
2. Les suites Oracle

2. http ://bpmstech.blogspot.fr/2011/05/lombardi-best-practices.html)

3. SAP NetWeaver
4. Quelques outils du monde du logiciel libre Open Source

a) IBM webSphere

Le logiciel IBM webSphere [3] pour les environnements SOA permet l'exécution des processus métiers dynamiques et inter-connectés, tout en offrant des infrastructures applicatives à hautes performances. Ces dernières sont adaptées à toutes les situations rencontrées par les entreprises [150].
Les différentes versions existantes, telles que : webSphere Business Compass, Business Modeler Advanced ou Rational Application Developer sont des outils web collaboratifs offrant des interfaces ergonomiques pour la conception et le développement des processus métiers. Dans ces environnements intégrés, les utilisateurs peuvent afficher et commenter les brouillons des processus publiés, et peuvent collaborer avec différents participants pour définir de bonnes pratiques en matière de modèles. Reposant sur l'édition webSphere Business Modeler Basic, la version Business Modeler Advanced permet aux professionnels de générer la modélisation, la simulation et l'analyse des processus métiers pour mieux comprendre, documenter et déployer leurs activités. webSphere Business Modeler Advanced aide à accélérer le développement des processus en vue d'une adaptation aux besoins métier évolutifs actuels [150].

D'une manière générale, les outils présentés par IBM aident les développeurs à concevoir, développer, assembler, tester et déployer des services web. Ils intègrent des outils de test et d'analyse, qui permettent d'identifier et de corriger les problèmes d'évolution. Cependant, des insuffisances sont constatées pour la gestion de la migration des instances actives. En effet, cette dernière est laissée à la charge de l'utilisateur qui est contraint de prendre des décisions, au cas par cas, pour trancher sur la poursuite des activités quand une nouvelle version du processus métier est déployée.

b) Les suites Oracle

Les outils fournis par Oracle, avec leur différentes versions : Oracle Business Process Management Suite, Oracle BPEL Process Manager et BPEL Designer sont des suites logicielles intégrées conçues pour modéliser, automatiser, gérer, simuler, optimiser et exécuter les processus métiers, les systèmes et les applications d'une organisation. Ces suites améliorent l'efficacité et la qualité des processus métiers en renforçant leur utilisation et leurs débits.

A titre d'illustration, l'outil Oracle BPEL Process Manager permet la réalisation de l'orchestration des processus métiers et des services web, par la mise en place d'un mécanisme d'intégration des éléments existants dans le système. Dans de tels environnements, les services informatiques peuvent déployer des processus opérationnels critiques gérant d'importants volumes, tout en s'appuyant sur les services de l'infrastructure existante [151]. Dans la version BPEL Designer, les utilisateurs peuvent modéliser via une interface graphique leurs processus métiers, en utilisant la spécification du langage BPEL [54]. Ceci garantit la portabilité des définitions des processus

3. https ://www-01.ibm.com/software/websphere/subscriptionandsupport/business-process-manager/versions/

métiers. De plus, les développeurs peuvent consulter et modifier directement le code source BPEL. Cet environnement est l'outil de référence des développeurs pour l'implémentation des processus métiers [151].

A noter que dans ces environnements, la gestion des différentes versions des processus métiers et des instances qui leur sont rattachées se font d'une manière quasi manuelle qui implique un effort particulier de la part de l'utilisateur. En plus, la question de la migration des instances n'a pas été pris en compte d'une manière efficace.

c) SAP NetWeaver

En sa qualité d'intégrateur d'applications d'entreprise, SAP [4] a élaboré une série d'outils pour la gestion du cycle de vie des processus métiers.

La plateforme SAP NetWeaver facilite l'intégration des outils en dépassant les frontières organisationnelles et technologiques, et elle réduit le coût de développement et de déploiement des processus métiers [152].

L'environnement SAP Netweaver Business Process Management procure aux spécialistes de la gestion et de l'informatique un environnement partagé pour concevoir, modéliser et exécuter des processus de gestion nouveaux ou adaptés, sans créer de code. Il facilite l'incorporation de règles de gestion claires dans les processus métiers. La modélisation des processus métiers avec les vues d'implémentation SAP assure la correspondance avec la stratégie de l'entreprise, ainsi que le support des objectifs de l'entreprise [152].

L'outil Mega SolMan présente une interface bidirectionnelle qui assure la synchronisation des modèles SAP avec les processus métiers. Cet outil réduit le temps et le coût d'un projet SAP, en identifiant en amont les besoins réels de l'organisation [153].

Il est constaté que dans les environnements concurrentiels actuels, où l'agilité et l'adaptabilité sont des impératifs, les solutions proposées par SAP ne sont pas en capacité d'apporter une plus value pour la prise en charge des changements. En effet, les entreprises modernes ont besoin d'une plateforme unique et flexible qui intègre, en plus de l'ensemble des collaborateurs et des informations manipulées par les processus métiers, les éventuelles évolutions et la gestion de la migration des instances de ces processus métiers.

d) Les logiciels libre (outils Open Source)

Certains outils *Open Source* offrent des solutions applicatives exploitables pour un usage professionnel en entreprise. L'avantage est que ces outils couvrent des besoins en matière de gestion, et restent ouverts aux clients qui peuvent les améliorer pour les adapter à leurs besoins spécifiques. Les outils les plus populaires sont : Bonita BPM [154], ProcessMaker [155] et Modelio [5] de Modeliosoft [156].

Bonita BPM Community est un outil de gestion des processus métiers (BPM) *Open Source* qui permet de modéliser les différentes étapes d'un processus métier, avec la prise en charge des tâches automatiques, des actions manuelles et des formulaires pour la saisie ou la validation des informations. L'outil intègre, aussi, la gestion des différents

4. **SAP** : Systems, Application and Products for data processing
5. https ://www.modelio.org/

groupes et les rôles des utilisateurs. Les utilisateurs finaux l'exploitent pour créer, suivre et réaliser les différentes tâches, activités et évènements liés à leur processus métiers [154].

ProcessMaker [155] est un outil efficace et facile à utiliser pour la gestion des processus métiers ou pour les applications logicielles de workflow. Il est utile aux organisations de toutes tailles, pour la conception, l'automatisation et le déploiement des processus métiers ou des workflows de toutes sortes. ProcessMaker est extrêmement efficace, léger et possède les meilleurs atouts de tous les logiciels de workflow dans l'industrie [155].

L'éditeur Modeliosoft a met sa nouvelle version de son logiciel phare, Modelio pour la modélisation des processus métiers. Cet outil bénéficie de l'engagement et de la diversité de la communauté de développeurs, de quoi favoriser les innovations dans le monde de la modélisation des processus métiers [156].

A signaler l'existence dans le domaine industriel d'une grande variété d'autres outils en code libre qui offrent la modélisation, l'automatisation, l'exécution et la supervision des processus métiers. Néanmoins, la question de la gestion des changements et de leurs impacts n'a été intégrée dans aucun outil.

4.4.2 Synthèse sur les outils logiciels du monde industriel

La question de la migration des instances actives n'a pas été prise en compte par une grande partie des fournisseurs de logiciels et le peu d'outils qui ont abordé ce problème proposent des solutions manuelles. Effectivement, certains industriels ont étoffé leurs suites BPM avec des outils, souvent graphiques, qui permettent de gérer la migration des instances actives lors de l'évolution des processus métiers. La solution la plus fréquente consiste à proposer des assistants graphiques qui permettent à un utilisateur expert de spécifier manuellement comment les instances actives sont transférées vers la nouvelle version du processus métiers.

Nous estimons que cette façon de faire reste, extrêmement, fastidieuse pour les gestionnaires des protocoles. En effet, la complexité de la tâche croit exponentiellement avec le nombre d'instances à faire migrer. Ce dernier est très considérable dans le contexte des services web mettant en œuvre des applications grand-public, telles que les domaines du : e-commerce, e-learning, e-gouvernement, ... De plus, la majorité des outils existants limite les possibilités de migration des instances (*e.g., seules les instances qui n'ont pas encore atteint les changements peuvent migrer*). En définitif, les suites BPM actuelles sont encore très basiques et la migration des instances actives reste une tâche ardue et non évidente.

4.5 Conclusion

La gestion de l'évolution des protocoles des services web, durant sa phase opérationnelle (*quand des instances sont en cours d'exécution*), est une tâche complexe. Dans ce chapitre, nous avons montrés que malgré la diversité des approches existantes, celles ci souffrent de certaines limites et ne répondent pas d'une manière suffisante à cette problématique. Ce constat est valable aussi bien pour les travaux du domaine de la recherche académique que pour celui des produits industriels.

En résumé les approches existantes pour la gestion de la migration forcent les instances actives à refléter, le plus fidèlement possible, celles ayant démarré leur exécution dans le protocole d'origine (*les instances d'origine*). En effet, les instances actives sont classées sur la base de leur historique des exécutions, et sont qualifiées de : *migrable* ou *non migrables*. En plus, dans ces approches il est impératif de cerner avec exactitude les instances non affectées par les changements, et qui seront les seules candidates à la migration (*c.f., état de l'art 4.3*).

Il ressort à partir de l'exploration approfondie des différents travaux existants dans les domaines connexes, que de nouveaux concepts et de nouveaux mécanismes sont plus que nécessaires pour remédier aux insuffisances constatées. Dans cette optique, et afin de pouvoir apporter des améliorations qualitatives, le problème de l'analyse de l'impact des évolutions doit être perçu dans le cadre d'une nouvelle approche globale qui prendra en charge la dimension modélisation à base de formalismes solides, et tout en intégrant les propriétés et les spécificités associées à la nature des évolutions des protocoles des services web.

Dans le chapitre prochain, nous allons présenter le premier apport de notre contribution qui consiste à spécifier un langage déclaratif pour la gestion de la migration des instances actives des services web.

Deuxième partie

Une approche déclarative pour la gestion de l'évolution dynamique des protocoles des services web

Chapitre 5

Un langage déclaratif de migration d'instances

5.1 Introduction

La contribution de cette thèse vise à proposer une nouvelle approche qui apporte des enrichissements aux travaux de recherche existants dans le domaine de l'analyse de l'impact des évolutions des protocoles des services web. Pour des raisons de clarté de l'exposé, nous procédons en deux étapes. Premièrement, nous exposons dans ce chapitre le cadre formel qui supporte l'approche proposée, et nous formalisons le langage de migration d'instances. La deuxième étape, reportée au prochain chapitre (*voir chapitre 6*), traite de l'extension du langage proposé par des patrons de migration.

Dans ce chapitre, nous exposons les ingrédients de base nécessaires à la spécification du langage de migration d'instances qui supporte notre approche, et nous explicitons sa formalisation. Nous commençons le chapitre par la section 5.2 qui décrit et illustre le modèle choisi pour la représentation des protocoles des services web. Après, nous introduisons dans la section 5.3 les concepts et les notations associés au modèle de protocole employé, et qui seront utilisés dans la suite du manuscrit. Dans la section 5.4, nous identifions un ensemble de critères pertinents pour la modélisation du problème, et nous y exposons quelques exemples de stratégies de migration relatives au contexte des évolutions dynamiques. Puis dans la section 5.5, nous présentons et nous décrivons le principe de fonctionnement de notre approche pour la gestion de la migration des instances actives. La section 5.6 est dédiée à la spécification du langage de migration d'instances, et nous clôturons le chapitre par la section 5.7 qui comprend une conclusion et des discussions sur le langage de migration proposé.

5.2 Choix du modèle de protocole de service

Différents modèles de protocoles métiers, dotés de niveaux d'expressivité variés, ont été proposés dans la littérature pour exprimer différents types de contraintes et d'abstractions décrivant les propriétés afférentes aux comportements des services web (*e.g., contraintes temporelles et transactionnelles* [10, 62]).

Dans nos travaux, nous utilisons une version de base du modèle de protocole métier des services web qui permet de décrire, essentiellement, les contraintes d'ordre régissant l'exécution des ***activités*** *(abstraites)* fournies par un service [10, 50, 157]. Un tel modèle est très adéquat pour répondre à notre problématique, car il permet d'analyser les instances en cours d'exécution, et il décrit le flux d'activités manipulées *(échange de messages, exécution d'une tâche, état atteint par une instance . . .)*. Nous rappelons ci-dessous la définition d'un tel modèle.

Définition 5.1 ***Protocole métier d'un service web*** *[11]*
Un protocole métier d'un service web est un tuple $\mathcal{P} = (Q, q_0, \mathcal{F}, \mathcal{M}, \mathcal{R})$*, tels que :*

- Q *est un ensemble fini d'états ;*
- $q_0 \in Q$ *est l'état initial du protocole ;*
- $\mathcal{F} \subseteq Q$ *est l'ensemble des états finaux du protocole ;*
- $\mathcal{M}$ *est un ensemble fini d'activités abstraites ;*
- $\mathcal{R} \subseteq Q \times Q \times \mathcal{M}$ *est une relation de transition. Chaque élément* $(q, q', m) \in \mathcal{R}$ *représente une transition d'un état source* q *vers un autre état cible* q'*, suite à l'exécution de l'activité* m*.*

Ainsi, un protocole métier d'un service web (*plus simplement,* ***protocole de service***) est formellement décrit par une machine à états finis déterministe (**FSM**, où les états représentent les différentes phases qu'un service peut parcourir pendant son exécution et les transitions représentent les activités qu'un service peut réaliser [11].

5.2.1 Motivations du choix du modèle de protocole

Le choix d'une machine à états finis pour représenter les protocoles métiers des services web est motivé par le rôle important joué par ce formalisme pour représenter les comportements des systèmes dynamiques. En effet, les machines à états finis constituent un des formalismes les plus utilisés dans la théorie des processus [158]. Elles ont, également, été intensivement utilisées dans le domaine de la gestion des processus métier (**BPM**) et celui des services web pour supporter l'analyse formelle des processus métiers [95]. En parallèle, différentes approches et techniques ont été proposées dans la littérature pour transformer divers modèles de protocoles vers des automates [92]. A titre d'illustration, certains travaux ont largement abordé les techniques de transformation des réseaux de Petri [93], des diagrammes UML [94] ou des programmes BPMN [90, 91] en des représentations sous forme d'automates. Par ailleurs, au delà de leur richesse sémantique et leur formalisme rigoureux, les automates sont reconnus par leur simplicité, leur popularité et l'existence d'une grande variété d'outils logiciels permettant leur spécification et leur vérification.

5.2.2 Un exemple de protocole de service

La figure 5.1, montre un exemple de protocole qui décrit un processus métier réaliste (*mais simplifié*) de gestion des dossiers de retraite. Ce protocole Retraite peut être, par exemple, déployé par un réseau national de caisses de sécurité sociale (CNR) [1], afin de prendre en charge les demandes des citoyens relatives aux prestations de retraite et de gérer, par la suite, le paiement des échéances périodiques aux profits des bénéficiaires. Dans un tel modèle de protocole, les noms des états (e.g : Demandeur Identifié, Rejet, Paiement activé, ...) sont des symboles sans signification et n'affectent pas l'usage opérationnel du service. Ils correspondent aux différentes étapes par lesquelles transite une demande de retraite. Par contre, les noms des transitions entre états (e.g : Contrôle Carrière, Notification Rejet,...) sont significatifs et représentent les activités qu'il faut invoquer pour passer d'une étape à une autre.

1. **C**aisse **N**ationale des **R**etraites.

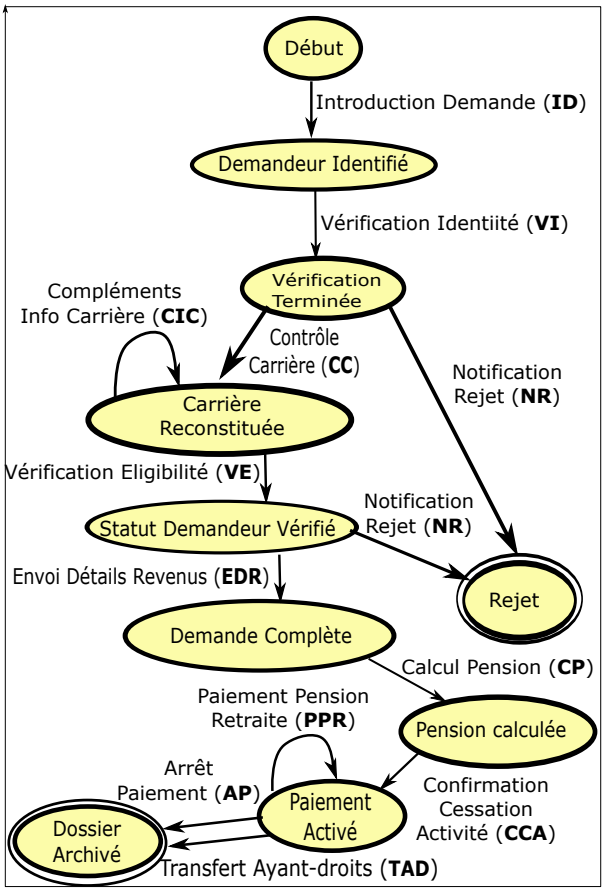

FIGURE 5.1 – Protocole métier du service gestion des retraites (**Retraite**).

Pour alléger la manipulation des noms des activités, ils sont abrégés par la suite comme indiqué dans la figure 5.1 (*e.g : l'activité* **V**érification **I**dentité est abrégée en **VI** et l'activité **C**onfirmation **C**essation **A**ctivité est abrégé en **CCA**,...).

Dans la pratique, un tel service web est, souvent, invoqué par plusieurs clients en même temps, et il est fréquent d'avoir un nombre important d'instances d'un même service qui sont en cours d'exécution simultanément. De même, comme dans beaucoup d'autres applications, les instances du service Retraite précédent, peuvent avoir des durées de vie relativement longues.

5.2.3 Description du protocole retraite

Initialement, quant un utilisateur accède à l'application, une nouvelle instance du service Retraite est créée à l'état Début. Ensuite, quand l'utilisateur remplit et soumet le formulaire de demande (*activité :* Introduction Demande), l'instance transite vers l'état Demandeur Identifié. A cette étape, la caisse des retraites opère des vérifications concernant l'identité du demandeur (*activité :* Vérification Identité). Il s'agit de s'assurer, par exemple, que le numéro de sécurité sociale est correct, que l'âge du demandeur est supérieur au minimum requis et que la nationalité ou le statut légal d'étranger sont valides. Après l'exécution de cette activité, l'instance du service passe à l'état Vérification Terminée. Si la vérification échoue, une notification de rejet est

adressée au demandeur (*activité :* Notification Rejet), et l'instance transite à l'état final Rejet avant de se terminer. Si, par contre, la vérification est achevée avec succès, l'instance exécute l'activité Contrôle Carrière, suite à laquelle elle passe à l'état Carrière Reconstituée. A ce stade, des informations supplémentaires peuvent être fournies par l'utilisateur. Elles seront traitées de manière itérative afin d'actualiser l'historique de carrière du demandeur (*activité :* Complément Info Carrière). Après la reconstruction de la carrière du demandeur, une vérification de son éligibilité est nécessaire (*activité :* Vérification Eligibilité). Cette activité permet de déplacer l'instance à l'état Statut Demandeur Vérifié. Ensuite, si pour diverses raisons, le demandeur n'est pas éligible pour bénéficier des avantages de retraite, un rejet lui est notifié (*activité :* Notification Rejet). Dans ce cas, l'instance passe à l'état final Rejet et se termine. Dans le cas où le demandeur est éligible pour un avantage de retraite, il lui est demandé de fournir des informations additionnelles concernant ses revenus qui vont servir de base pour le calcul du montant de la pension à servir (*activité :* Envoi Détails Revenus). Cette activité fait déplacer l'instance à l'état Demande Complète, à partir duquel le calcul du montant de la pension peut être effectué (*activité :* Calcul Pension), tout en transférant l'instance vers l'état Pension Calculée. A cette étape, si le demandeur est intéressé, il transmis son certificat de cessation d'activité (*activité :* Confirmation Cessation Activité) pour activer la mise en payement de sa pension. Cette dernière activité déplace l'instance du service à l'état Paiement Activé, où une activité Paiement Pension Retraite est exécutée périodiquement (*chaque mois, par exemple*). Finalement, l'instance peut passer de l'état Paiement Activé à l'état final Dossier Archivé, où elle termine son exécution dans deux cas de figures distincts : soit, l'annulation du paiement des avantages (*activité :* Arrêt Paiement) pour différentes raisons (*par exemple, certificat de vie non fourni*), soit le transfert de la pension aux ayant-droits (*activité* Transfert Ayant-droits).

Il convient de signaler que le protocole métier présenté ci-dessus est, en fait, une version simplifiée de celui qui gère réellement les procédures de la retraite. Néanmoins, il est suffisant pour illustrer notre approche. En effet, un processus réel de gestion de la retraite est beaucoup plus complexe et intègre, généralement, plusieurs règles de gestion et des procédures spécifiques (*retraite proportionnelle, retraite sans conditions d'age, régime agricole, régime des non salariés,...*) et chacune des options est gouvernée par des règles métier spécifiques et complexes.

D'autre part, les instances du protocole précédent sont généralement de longue durée pour deux raisons principales : (i) le traitement d'un dossier de retraite peut prendre plusieurs mois, et (ii) en raison de l'accroissement de l'espérance de vie de la population, la durée passée à la retraite est beaucoup plus longue. Comme conséquence directe, ils très fréquent de trouver à un instant donné des dizaines de millions d'instances de notre service Retraite de la figure 5.1 qui sont à l'état active (Paiement Activé) pendant plusieurs années. [2]

2. Par exemple, en Algérie le nombre de salariés retraités de droit direct ou dérivé jusqu'à Juin 2016 s'élève à : **2 880 180** (http ://cnr-dz.com/chiffres-caracteristiques/ (*consulté le 24/12/2016*).

5.3 Notions associées au modèle de protocole

Nous introduisons, dans ce qui suit, les notions de base, les définitions utiles pour notre analyse et les notations qui seront utilisées dans la suite du manuscrit.

Nous notons par $\Sigma = \{\epsilon, a, b, c, d, e, f, g, \ldots\}$ l'alphabet qui représente l'ensemble des symboles correspondant aux noms des *activités* fournies par un service web. Ces symboles sont associés aux opérations (*les activités abstraites*) et expriment les différentes transitions du service d'un état vers un autre.
La chaine vide $\epsilon \in \Sigma$ dénote la transition nulle.

Par exemple, dans le protocole Retraite de la figure 5.1, les activités sont exprimées par l'ensemble de symboles :
$\Sigma = \{\epsilon, ID, VI, NR, CC, CIC, VE, EDR, CP, CCA, PPR, AP, TAD\}$.

Nous commençons par rappeler la définition du langage d'un automate.

Définition 5.2 ***Langage d'un automate***
On considère les machines d'états finis déterministes comme des automates, et on définit le langage associé, noté L($\mathcal{P}$), comme étant l'ensemble des mots acceptés (reconnus) par l'automate $\mathcal{P}$. Une chaine (mot) est accepté par l'automate s'il existe un chemin dans l'automate en question, qui commence par l'état initial, qui se termine à l'un des états finaux et qui reconnu ce mot symbole par symbole [159].

Nous introduisons, à présent, le concept d'exécution d'un protocole

Définition 5.3 ***Exécution et chemin d'exécution d'un protocole***
Étant donné un protocole $\mathcal{P} = (Q, q_0, \mathcal{F}, \mathcal{M}, \mathcal{R})$, une ***exécution*** *de $\mathcal{P}$, notée* e*, est une séquence alternée d'états et d'activités de $\mathcal{P}$, telle que :*
e$=q_0.m_0.q_1 \ldots q_n.m_n.q_{n+1}$*, qui vérifie les trois conditions suivantes : (i) débute par l'état initial q_0 de P, (ii) se termine par un état q_{n+1} de $\mathcal{P}$, et (iii) respecte la relation de transition de $\mathcal{P}$, i.e., $(q_i, q_{i+1}, m_i) \in \mathcal{R}, \forall i \in [0, n]$.*

Une exécution e est dite **complète**, si elle commence par un état initial (*i.e.,* q_0) et se termine par un état final (*i.e., si* $q_{n+1} \in \mathcal{F}$).

A toute exécution $\mathsf{e} = q_0.m_0.q_1.m_1 \ldots q_l.m_l \ldots q_n.m_n.q_{n+1}$ de $\mathcal{P}$ est associée un **chemin d'exécution** de $\mathcal{P}$, noté c, tel que $\mathsf{c} = m_0.m_1 \ldots m_l \ldots m_n$, constitué seulement de la séquence finie d'activités obtenue suite à la suppression des noms des états de l'exécution e. Un chemin d'exécution est dit **complet** s'il est issu d'une exécution complète.

Le concept de chemin d'exécution est très utile pour distinguer les types d'instances de service. En effet, les chemins d'exécution complets correspondent à des instances qui sont déjà terminées, alors que les chemins non complets reflètent des instances qui sont encore en cours d'exécution, ou **instances actives**.

Exemple 5.1 ***Exécutions et chemins d'exécution d'un protocole***
A titre d'exemples, les trois expressions suivantes :
e_1=*debut.ID.Demandeur Identifié.VI.Vérification terminée.NR.Rejet,*
e_2=*debut.ID.Demandeur Identifié.VI.Vérification terminée.CC.Carrière Reconstituée.VE.Statut Demandeur Vérifié, et*
e_3=*debut.ID.Demandeur Identifié.VI.Vérification terminée.CC.Carrière Reconstituée.CIC.Carrière*

Reconstituée.VE.Statut Demandeur Vérifié.EDR.Demande Complète,
sont des exécutions du protocole Retraite *de la figure 5.1.*

On observe que e_1 *est la seule exécution complète. Le chemin d'exécution* $\mathsf{c}_1 = ID.VI.NR$ *associé à* e_1 *est complet, car il permet d'atteindre l'état final* Rejet $\in \mathcal{F}$.

Le langage $L(\mathcal{P})$ reconnu par l'automate $\mathcal{P}$ permet d'identifier l'ensemble de tous les chemins d'exécution complets de $\mathcal{P}$. Afin de permettre, aussi, la manipulation des chemins d'exécution incomplets, on dénote par $Exec(\mathcal{P})$ l'ensemble de tous les chemins d'exécution possibles du protocole $\mathcal{P}$.

Lors de l'invocation d'un service par les clients, les instances associées empruntent divers chemins d'exécution en invoquant les différentes activités, et tout en générant des traces d'exécution.

Nous définissons, ci-après la notion de **trace ou historique d'exécution**.

Soit i une instance d'un protocole $\mathcal{P}$, et soit $\mathsf{e} = q_0.m_0.q_1 \ldots q_n.m_n.q_{n+1}$ l'exécution de i dans $\mathcal{P}$.

Définition 5.4 ***Trace d'exécution d'une instance de service***
La ***trace d'exécution*** *ou* ***historique d'exécution*** δ *d'une instance* i *du protocole* $\mathcal{P}$ *est exprimée par son chemin d'exécution* $\mathsf{c} = m_0.m_1 \ldots m_n$*, extrait de l'exécution* e*. Elle représente la séquence finie des activités historiques qui ont été exécutées par l'instance* i*, depuis le début de l'invocation du service jusqu'à l'état courant de* i*. Une trace d'exécution est dite* ***complète*** *si elle est issue d'un chemin d'exécution complet.*

A noter que différentes instances d'exécution peuvent être exprimées par la même trace ; *i.e le même chemin d'exécution.* Cependant, chaque instance est caractérisée par d'autres attributs descriptifs qui lui sont spécifiques, par exemple : *l'identifiant de l'instance, l'utilisateur, les données manipulées, le temps de démarrage, ...*

A titre d'illustration, nous donnons ci-après trois traces d'exécution associées à des instances d'exécutions différentes du protocole Retraite de la figure 5.1. Ces traces sont issues, respectivement, des chemins d'exécutions e_1, e_2 et e_3 de l'exemple 5.1.

δ = ID.VI.NR
α = ID.VI.CC.VE
β = ID.VI.CC.CIC.VE.EDR

Parmi ces traces, seule δ est une trace d'exécution complète, car son chemin d'exécution e_1 est complet, alors que α et β sont des traces incomplètes et représentent des instances en cours d'exécutions.

L'ensemble des traces d'exécution complètes possibles d'un automate $\mathcal{P}$ est donné par le langage $L(\mathcal{P})$ reconnu par $\mathcal{P}$ (*i.e., l'ensemble des mots acceptés par* $\mathcal{P}$).

Pour toute trace d'exécution δ d'un service, nous utiliserons également les notations et définitions de base suivantes :

- $|\delta|$ dénote la longueur de la trace d'exécution δ, i.e : le nombre d'activités figurant dans δ. Par exemple, pour les traces d'exécution précédentes, nous avons : $|\delta| = 3$, $|\alpha| = 4$ et $|\beta| = 6$
- Une **sous-trace** $\delta' = st(l, i, \delta)$ de δ est une séquence d'activités de δ qui commence par δ_i, et dont la taille est l. Par exemple, $\beta' = st(2, 3, \beta) = CC.CIC$
- Un **préfixe d'une trace d'exécution**, noté $\omega = Prefixe(m, \delta)$, est une sous-trace, telle que $\delta = \omega.\delta'$, avec $|\omega| = m$.
  Par exemple, $\omega = Prefixe(3, \beta) = ID.VI.CC$.

- Un **suffixe d'une trace d'exécution** $\omega = Suffixe(n, \delta)$ est une sous-trace de δ, telle que $\delta = \delta'.\omega$, avec $|\omega| = n$.
  Par exemple, $\omega = Suffixe(2, \beta) = VE.EDR$
- **L'opérateur de Kleene, noté * :** Appliquée à un alphabet Σ (*ensemble de symboles*), l'opérateur a pour résultat le langage Σ^*, constitué par l'ensemble des mots sur Σ, mot vide ε inclus [159]. Par exemple : $\{a, b\}^*$ donne le langage : $\{\varepsilon, a, b, ab, ba, aa, bb \ldots\}$, et $\{a\}^*$ donne le langage : $\{\varepsilon, a, aa, aaa, aaaa \ldots\}$.
- **L'opérateur Shuffle :** L'application de l'opérateur Shuffle [160] à une trace d'exécution δ, noté $\otimes(\delta)$, renvoie le langage *(ensemble de traces d'exécution)* composé de tous les mots possibles obtenus par la permutation des activités de δ. Par exemple, $\otimes(abc) = \{abc, acb, bca, bac, cab, cba\}$. Pour la trace d'exécution δ précédente, nous avons :
  $\otimes(\delta) = \{ID.VI.NR, ID.NR.VI, VI.ID.NR, VI.NR.ID, NR.ID.VI, NR.VI.ID\}$.

Nous introduisons, à présent, la notion de relation de simulation entre protocoles qui est très utile dans notre contexte [11, 49, 157].

Définition 5.5 ***Relation de simulation entre protocoles*** *[49]*
Soient $\mathcal{P} = (Q, q_0, \mathcal{F}, \mathcal{M}, \mathcal{R})$ *et* $\mathcal{P}' = (Q', q'_0, \mathcal{F}', \mathcal{M}', \mathcal{R}')$ *deux protocoles de service. Un état* $q_1 \in Q$ *est simulé par un état* $q'_1 \in Q'$*, noté :* $q_1 \ll q'_1$*, si et seulement si les deux conditions suivantes sont vérifiées :*

- $\forall\ a \in \mathcal{M}$ *et* $\forall\ q_2 \in Q$ *avec :*$(q_1, q_2, a) \in \mathcal{R}$ *, il existe une transition* $(q'_1, q'_2, a) \in \mathcal{R}'$*, telle que :* $q_2 \ll q'_2$ *;*
- *Si* $q_1 \in \mathcal{F}$*, alors* $q'_1 \in \mathcal{F}'$ *;*
  On dit que $\mathcal{P}$ *est simulée par* $\mathcal{P}'$*, et on note :* $\mathcal{P} \ll \mathcal{P}'$*, si :* $q_0 \ll q'_0$

Exemple 5.2 *Simulation de protocoles [49]*

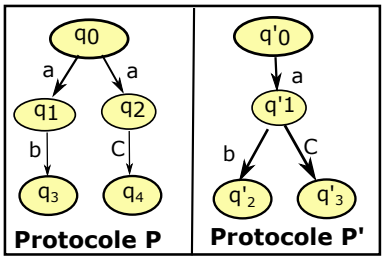

FIGURE 5.2 – Relation de simulation entre deux protocoles de services

Dans cet exemple, le protocole $\mathcal{P}'$ peut reproduire le comportement du protocole $\mathcal{P}$, mais pas le contraire. En effet, toutes les transitions du protocole $\mathcal{P}$ sont réalisables par le protocole $\mathcal{P}'$. Cependant, l'inverse est faux. C'est le cas, par exemple, lorsque le protocole $\mathcal{P}'$ est à l'état q'_1 et exécute l'activité c alors que le protocole $\mathcal{P}$ est à l'état q_1. Dans ce cas, $\mathcal{P}$ n'a de choix possibles que d'exécuter l'activité b. Donc, il ne reproduit pas le comportement prévu par le protocole $\mathcal{P}$. Par conséquent, le protocole $\mathcal{P}'$ **simule** le protocole $\mathcal{P}$, par contre protocole $\mathcal{P}$ ne simule pas le protocole $\mathcal{P}'$.

La notion de simulation entre protocoles est très utile dans notre contexte, car elle permet de comparer deux protocoles entre eux sur la base de leur comportement.

Pour terminer cette section, nous introduisons la notion de sous-protocole qui est très utile pour nos développements futurs.

Définition 5.6 ***Spécification d'un sous-protocole***
Soit $\mathcal{P} = (Q, q_0, \mathcal{F}, \mathcal{M}, \mathcal{R})$ un protocole de service, et soit $q \in Q$ un état de $\mathcal{P}$. Le sous-protocole $\mathcal{C}_{\mathcal{SP}}(\mathcal{P}, q)$ de $\mathcal{P}$ est le protocole : $\mathcal{C}_{\mathcal{SP}}(\mathcal{P}, q) = (Q'_s, q'_{0s}, \mathcal{F}'_S, \mathcal{M}'_S, \mathcal{R}'_S)$, obtenu à partir de $\mathcal{P}$ comme suit :

- *q est l'état initial de $\mathcal{C}_{\mathcal{SP}}(\mathcal{P}, q)$,*
- *$Q'_s \subseteq Q$ contient l'ensemble des états de Q accessibles à partir de q en utilisant la relation $\mathcal{R}$ de $\mathcal{P}$,*
- *$\mathcal{F}'_s = Q'_s \cap \mathcal{F}$,*
- *$\mathcal{R}'_s = \{(q, q', m) \in \mathcal{R} \ \ s.t. \ \{q, q'\} \subseteq Q'_s\}$,*
- *$\mathcal{M}'_s \subset \mathcal{M}$ est constitué des messages $\mathcal{M}$ qui apparaissent dans les transitions de $\mathcal{R}'_s$.*

Conformément à cette formalisation, le sous-protocole $\mathcal{C}_{\mathcal{SP}}(\mathcal{P}, q)$ permet de capturer toutes les exécutions possibles de $\mathcal{P}$ à partir de l'état q. En fait, il spécifie tous les chemins d'exécution futurs de $\mathcal{P}$ qui commencent par l'état q et qui se terminent à l'un des états finaux de $\mathcal{P}$.

En se référant au protocole Retraite de la figure **5.1**, la notion de sous- protocole peut être illustrée en choisissant l'état Statut Demandeur Vérifié de la figure **5.1**. Le sous-protocole obtenu : $\mathcal{C}_{\mathcal{SP}}$ (Retraite, Statut Demandeur Vérifié) commence par l'état Statut Demandeur Vérifié et intègre les trois chemins d'exécution complets suivants : $\{\alpha_1, \alpha_2, \alpha_3\}$, avec :

α_1 = EDR.CP.CCA. (PPR)*. AP
α_2 = EDR.CP.CCA. (PPR)*. TAD
α_3 = NR

5.4 Caractérisation des changements des protocoles

L'un des facteurs clés de performance de l'entreprise moderne réside dans l'agilité de son système d'informations (**S.I**). C'est à dire, la capacité de ce système à refléter, à tout moment, la réalité de l'organisation. En effet, le S.I doit, impérativement, gérer les changements qui surviennent dans l'environnement de l'entreprise. En se sens, la capacité des organisations modernes à s'**adapter aux changements** est un besoin stratégique réel qui leur garantit la survie dans des systèmes économiques, fortement, caractérisés par la concurrence rude et les coalitions conjoncturelles.

Les besoins d'adaptation passent, fondamentalement, par la gestion des changements des processus métiers. Dans le contexte des services web, ces changements se traduisent par des modifications qui peuvent affecter les interfaces de services et/ou les paramètres de qualité de service, aussi bien que la description des protocoles de services. La mise en œuvre des changements qui surgissent induit des impacts directs sur les instances du service. Ces dernières auront, incontournablement, besoin de migrer pour s'adapter aux nouvelles exigences imposées par le nouveau protocole.

Dans notre approche nous nous intéressons à l'évolution des protocoles des services web, et nous visons les objectifs suivants :

- capturer les différences et les similitudes entre l'ancienne et la nouvelle version du protocole de service web,
- prendre en compte l'état d'exécution de chaque instance active en considérant sa trace d'exécution réelle (*chemin d'exécution emprunté*),
- analyser l'impact de l'évolution des protocoles de services sur les instances en cours, et décider de la manière dont les changements seront propagés.

Abstraction faite aux raisons qui ont produit les changements, répondre à de telles préoccupations est une tâche complexe qui dépend de plusieurs paramètres, dont les plus pertinents sont :

1. le modèle de description des protocoles de services ;
2. la nature du processus d'évolution ;
3. la nature des opérations de changement opérées ;
4. les stratégies de migration déployées pour gérer les changements.

La combinaison de ces considération avec les objectifs visés exige une approche globale qui permettra de spécifier diverses stratégies de migration capables d'opérer une analyse de l'impact des changements sur les instances en cours d'exécution.

En se basant sur le modèle de description des protocoles de service décrit précédentes (*c.f. 5.2 et 5.3*), dans cette section nous analysons ces paramètres (*2, 3 et 4*), et nous mettons en évidence les choix retenus dans le cadre de notre approche. Ces choix sont des critères déterminants pour la modélisation du problème de l'analyse de l'impact des évolutions des protocoles des services web.

5.4.1 La nature du processus d'évolution des protocoles

Généralement, deux types d'évolution sont distingués pour les différents domaines concernés par les changements (*génie-logiciel, bases de données, Workflows, services web, ...*), à savoir : **les évolutions statiques** et **les évolutions dynamiques**.

Dans le contexte particulier des services web, les évolutions statiques sont caractérisées par l'absence d'instances actives au moment précis du changement. Dans ce cas, la gestion de l'évolution du protocole de service se limite, tout simplement, au déploiement de la nouvelle version du protocole. Les nouvelles instances pourront, par la suite, démarrer leurs exécutions conformément aux nouvelles spécifications et aucune difficulté n'est rencontrée.

Cependant, dans le cadre des évolutions dynamiques, certaines instances sont encore en cours d'exécution au moment du changement du protocole. Ces instances dites **actives** ont engagé leurs conversations avec le fournisseur du service web sur la base d'un protocole qui a subi des modifications de sa description, alors que des instances sont en cours d'exécution.

Pour gérer la progression des instances actives dans le contexte des évolutions dynamiques des services web, deux solutions sont possibles.

- laisser toutes les instances actives poursuivre leur exécution selon l'ancien protocole de service ;
- annuler toutes les instances actives et les redémarrer conformément à la nouvelle spécification du protocole.

Cependant, ces deux solutions extrêmes sont rarement possibles en pratique pour des raisons de préservation de l'historique des exécutions et pour des besoins de mise en conformité immédiate avec les nouvelles règles métier qui doivent entrer en vigueur le plus tôt possible. Dans notre approche, on s'intéressera à l'évolution dynamique des protocoles, et nous proposons une approche qui permettra la continuité de l'exécution des instances actives dans le protocole évolué.

5.4.2 Le type d'opérations de changement considérées

Nous nous appuyons sur le modèle de protocole de service introduit dans la section 5.2, et nous considérons les évolutions des services qui se traduisent par des modifications de la description de leur protocole. Les opérations d'actualisation qui sont permises constituent une dimension importante du problème. Généralement, deux classes de changements sont distinguées :

- La modification de la description d'un élément contenu dans la spécification du protocole. Par exemple, la modification du nom d'une activité ou d'un état.
- La modification de la structure du protocole elle même, comme par exemple, l'ajout ou la suppression d'une activité.

Ces deux types d'opérations de base peuvent être composés, de manière à exprimer des changements plus complexes. Par exemple, la fusion des états, le déplacement des activités (*réorganisation*) ou l'ajout d'un sous-protocole en entier (c.f 3.3.1).

Si la première catégorie des changements est considérée comme assez facile à gérer, la seconde classe de changements est plus complexe, en particulier, dans le contexte des changements dynamiques. En effet, dans ce dernier cas, une analyse de l'impact des changements sur les instances actives doit être conduite afin d'identifier les actions nécessaires à déclencher pour rendre ces instances conformes aux nouvelles exigences imposées par le nouveau schéma du protocole de service.

5.4.3 Les stratégies de migration pour gérer les changements

Dans notre contexte, une stratégie de migration exprime un ensemble de propriétés utiles qu'il faut préserver lors du processus de migration des instances actives. Autrement dit, elle spécifie des conditions précises que les instances actives doivent satisfaire pour qu'elles puissent continuer leurs exécutions dans le nouveau protocole, tout en décrivant leurs configurations correspondantes dans le contexte de ce dernier. En réalité, les conditions à satisfaire expriment, des choix de gestion et des besoins métier du fournisseur de service. Dans cette optique, les stratégies de migration à définir doivent être explicitées d'une manière formelle et précise.

Dans [24], trois types de stratégies de migration ont étés explorées : *la continuité de l'exécution conformément à l'ancien protocole*, *la migration vers des protocoles ad-hoc* et *la migration vers le nouveau protocole*. Dans notre approche, nous nous focalisons sur la dernière catégorie de stratégies de migration compte tenu, à la fois, de son importance du point de vue pratique et de la difficulté de sa mise en œuvre.

D'une manière générale, définir une stratégie de migration est une tâche complexe qui dépend étroitement de nombreuses décisions d'entreprise [135]. Nous estimons qu'une stratégie de migration efficace doit être **sélective** et à **granularité fine**, dans

le sens où elle doit être capable de distinguer entre les différentes instances actives en s'appuyant sur des critères de sélection très précis.

Nous avons identifié deux axes d'analyse pertinents pour la définition des stratégies de migration des instances actives.

- Les exécutions **historiques vs. futures** : la stratégie de migration peut se concentrer sur les propriétés liées à des exécutions passées et/ou certaines propriétés des activités potentielles qui peuvent être effectuées dans le futur.
- Les contraintes **d'obligation vs. d'interdiction** : la stratégie de migration peut viser l'instauration de certaines actions obligatoires, comme par exemple l'exécution obligatoire d'une activité importante, ou elle peut cibler l'interdiction de certaines situations (*par exemple, interdiction d'une activité future*).

Nous exposons, ci-après, quelques exemples de stratégies de migration qui peuvent être déployées pour gérer la migration des instances actives, et nous illustrons leur utilité et leurs objectifs.

a. Stratégie maximisant la préservation de l'historique

Dans certains protocoles spécifiques, les activités exécutées par les instances durant leur historique, peuvent être importantes et couteuses. Alors, le gestionnaire de protocole vise à préserver les historiques des exécutions lors de la migration. En exploitant cette stratégie, l'objectif est d'éviter la perte de travail, tout en préservant au maximum les activités déjà effectuées dans le passé.

La stratégie de migration basée sur la préservation de l'historique des exécutions est recommandée lorsque les activités du protocole sont de longue durée et consommatrices de ressources (*réseau, temps machine, temps d'utilisation du service ...*).

b. Stratégie maximisant le nombre d'instances migrables

Cette stratégie projette d'assurer la migration du nombre maximum d'instances actives. Elle est utile lorsque le nombre d'instances est consistant, alors que les changements sont significatifs. Par conséquent, le gestionnaire de protocole vise à imposer immédiatement ces changements. C'est le cas, par exemple, lors de l'application des exigences de sécurité (*nouveau protocole d'authentification, par exemple*), ou lors de la mise en place d'une nouvelle réglementation (*une nouvelle procédure qui doit entrer en vigueur le plus tôt possible*).

c. Stratégie de migration basée sur l'analyse des traces d'exécution

Du fait que le nombre d'instances actives peut être très élevé, l'analyse de leur traces d'exécution peut orienter les gestionnaires dans le choix de la meilleure stratégie de migration à adopter. En fait, les informations relatives aux activités exécutées, aux sous-chemins parcourus et aux états atteints par les instances, sont des indicateurs potentiels qu'il faut prendre en considération lors de la spécification d'une stratégie de migration.

A titre d'exemple, le gestionnaire de protocole peut stipuler que les instances n'ayant pas encore atteint un état spécifique (*affecté par les changements*), ou celles n'ayant pas exécuté un sous-protocole particulier, ou encore les instances qui ne sont

pas en train d'exécuter un sous-chemin donné, sont les seules instances qui peuvent migrer vers le nouveau protocole.

d. Stratégie de migration basée sur les exécutions futures

Dans cette stratégie, l'accent est mis, particulièrement, sur les exécutions futures des instances actives. En effet, les gestionnaires veulent faire migrer les instances actives, tout en minimisant le nombre d'activités à exécuter durant les interactions futures (*pour atteindre un état final*).

Cette dernière stratégie est intéressante lorsque les contraintes de temps sont importantes, et si la terminaison des exécutions pour les instances en cours est la préoccupation majeure. Cette option de migration peut être adoptée dans les cas de changement de la nature de l'activité de l'entreprise, sa liquidation économique ou encore lors des rachats d'entreprises par d'autres, ...

A signaler que la liste précédente des stratégies de migration n'est pas exhaustive. En fait, d'autres stratégies peuvent être proposées afin de répondre à des besoins spécifiques et diversifiés des utilisateurs. Néanmoins, ces stratégies demeurent figées, car elles sont spécifiées d'une manière prédéfinie. Or, l'un des soucis permanent des gestionnaires de protocoles est de disposer d'outils efficaces leur permettant de spécifier d'une manière **flexible** des stratégies de migration **personnalisées**.

Pour répondre à cette préoccupation, nous proposons une approche déclarative basée sur un langage de migration d'instances qui permet aux fournisseurs de services de spécifier leurs propres stratégies de migration lors de l'évolution dynamique des protocoles des services web. Chaque stratégie à définir permettra de mettre l'accent sur une ou plusieurs propriétés à respecter lors de la migration des instances actives vers la nouvelle version du protocole de service.

5.5 Description de l'approche déclarative

Dans le contexte de l'évolution dynamique des protocoles des services web, les instances actives ne sont pas toujours en mesure de satisfaire les contraintes imposées par le nouveau protocole. Cette restriction est due, fondamentalement, au fait que ces instances ont commencé leurs exécutions sur la base de l'ancien schéma du protocole de service. Se pose, alors, la question de la gestion de la migration des instances actives vers le nouveau protocole, suite à l'entrée en vigueur des changements portés par le nouveau protocole.

Pour surmonter les limites des approches classiques de gestion de la migration, nous proposons une nouvelle approche qui donne la possibilité aux fournisseurs de services de définir, d'une **manière déclarative**, les contraintes qui gouvernent le processus de migration des instances actives. Ainsi, ils auront plus de marge de manœuvre pour spécifier des **stratégies de migration** adaptées à leurs besoins spécifiques. Cette façon d'appréhender le problème contribue, inévitablement, à la maitrise de l'analyse de l'impact du changement. En effet, ces stratégies de migration **flexibles** offrent plus de possibilités aux instances actives pour qu'elles puissent continuer leurs exécutions, conformément aux nouvelles spécifications du protocole ayant subi des changements.

5.5.1 Caractéristiques de l'approche

L'objectif est de proposer une approche de migration qui prend en considération les insuffisances décelées durant la phase d'analyse des travaux connexes du domaine (*chapitre 4*), et qui apporte des enrichissements fondamentaux par rapport aux approches existantes pour la gestion de la migration des instances actives des services web.

Nous proposons une approche qui est basée sur la perception de la migration des instances actives comme étant le problème de sélectionner des instances s'exécutant sous un protocole de service donné et de les **convertir** en de nouvelles instances qui seront conformes avec la spécification du nouveau protocole [161]. Les fournisseurs de services peuvent, ainsi, spécifier leurs propres stratégies de migration, indépendamment des contraintes imposées par la rigidité dictée par la mise en conformité du nouveau protocole avec l'ancien.

Le trait saillant de notre approche consiste en un cadre formel fondé sur **un langage déclaratif** pour assister les fournisseurs de services dans la définition des stratégies de migration à granularité fine des instances actives. Ainsi, au lieu de forcer les instances actives à refléter le plus fidèlement possible celles d'origine, nous donnons aux fournisseurs de service la possibilité de spécifier, d'une manière déclarative, les contraintes qui gouvernent le processus de migration de ces instances.

Dans cette perspective, l'approche proposée est caractérisée par sa capacité de prendre en compte différentes dimensions du problème de migration, à savoir :

1. Une analyse de la compatibilité des protocoles dans une dimension plus flexible qui dépasse la simple vérification de la compatibilité stricte.
2. Au delà de la compatibilité historique, la migration des instances est considérée, aussi, dans sa portée future qui traite les exécutions des activités à venir.
3. Les opérations de changement qui peuvent affecter les structures du modèle du protocole de service sont complexes (*chemin, sous-chemin, sous-protocole, ...*)
4. En plus de la spécification des contraintes d'obligation, que les instances actives doivent satisfaire lors de la migration, d'autres types de contraintes sont prises en charge (*interdiction, migration conditionnelle,...*).
5. Afin de satisfaire différentes stratégies de migration spécifiques et diversifiées, la migration est abordée d'une manière formelle, consolidée par une assise théorique solide basée sur des modèles génériques.

5.5.2 Fondement de l'approche déclarative

Le cadre formel qui supporte l'approche proposée est fondé sur les deux éléments suivants.

– la spécification d'un **langage de migration d'instances** ;

– l'identification et la formalisation d'un ensemble de **patrons**[3] **de migration**.

Premièrement, le langage de migration d'instances permet de décrire des **règles de migration d'instances**. Une règle de migration spécifie les contraintes à satisfaire pour transférer une instance d'un protocole initial vers un nouveau protocole. Ainsi,

3. On utilisera le terme plus répandu **pattern**

elle permet de convertir les traces des instances choisies dans l'ancien protocole de service, vers des instances correspondantes dans le nouveau protocole.

Pour permettre la prise en charge des différents historiques d'exécution, tout en répondant à la large diversité des besoins de gestion, chaque règle de migration du langage proposé doit être sélective (*requête sur les traces*), et en même temps, elle doit être capable de répondre aux règles de gestion particulières (*les contraintes imposées par l'évolution*). Ainsi, chaque règle de migration, utilisée dans le cadre d'une stratégie de migration donnée, doit assurer les deux objectifs principaux suivants :

- permettre de sélectionner les instances actives qui vérifient des conditions particulières dans l'ancien protocole (*les pré-conditions*),
- convertir les instances sélectionnées en des instances du nouveau protocole. Une fois transférées, les traces des instances obtenues doivent vérifier les contraintes imposées par le nouveau protocole (*post-conditions*).

Deuxièmement, en plus des règles de migration, le langage déclaratif proposé est enrichi et étendu par un ensemble de patterns de migration. Les patterns de migration formalisés dans notre approche sont génériques, paramétrables et assurent différentes stratégies de migration des instances actives. Chaque pattern de migration se focalise sur un type particulier d'instances à faire migrer de l'ancien protocole vers le nouveau. Donc, chaque pattern exprime une règle générique de migration qui cible une catégorie spécifique d'instances actives à transférer.

En se basant sur les deux éléments précédents, dans notre approche, une stratégie de migration est exprimée par un ensemble de patterns de migration, où chaque pattern capture une classe de contraintes dépendantes de l'application, et il peut être instancié par le fournisseur de service pour définir des règles de migration spécifiques dans le cadre d'une stratégie de migration particulière.

Dans la prochaine section, nous présentons le mécanisme de fonctionnement du langage de migration d'instances, et nous décrivons sa formalisation. Après, dans le prochain chapitre (*chapitre 6*), nous identifions et nous formalisons l'ensemble des **patterns de migration**.

5.6 Spécification du langage de migration

Dans notre approche, une stratégie de migration se traduit par un ensemble de **règles** (*ou requêtes*) **de migration d'instances**. Chaque règle de migration doit être sélective et en mesure de prendre en charge des choix de gestion ciblés. Pour élaborer de telles stratégies, nous commençons par introduire le principe de fonctionnement qui met en action le langage de migration des instances actives des services web.

5.6.1 Principe du langage de migration d'instances

Tout d'abord, nous cernons les entrées de notre problème de migration d'instances comme étant les suivantes :

- une spécification d'un protocole de service source qui est représenté par un automate (*automate source*) ;
- les instances actives et leurs traces d'exécution (*des chemins d'exécution*) ;
- une nouvelle spécification du protocole de service (*automate cible*).

En se basant sur ces éléments, nous ramenons le problème de la migration d'instances à celui de la définition d'une **technique de transformation** [4] entre les structures des deux automates source et cible. Appréhender le problème en termes de transformation offre la possibilité de décrire des spécifications abstraites de haut niveau, qui permettent de définir les relations de correspondance entre les deux schémas de protocoles de service (*automates source et automate cible*). Dans cette perspective, on utilise des **règles de transformation** d'éléments descriptifs associés à une structure source, en des descriptions correspondantes dans une autre structure cible. De telles transformations sont destinées à être exploitées pour convertir les traces d'exécution des instances de l'automate source en des traces correspondantes dans l'automate cible [161]. En utilisant des techniques de mapping, le langage de migration à formaliser est en mesure de transformer un chemin d'exécution de l'automate source en un autre chemin correspondant dans l'automate cible, tout en exploitant des règles de migration spécifiques. Ces règles de migration définissent comment l'automate source est défini en termes d'existence de certains patterns.

Pour que le mapping à réaliser soit opérationnel, nous nous inspirons de l'approche classique utilisée dans le domaine de l'intégration de données, et qui est exploitée pour les échanges de données (*Data exchange*) [162]. Cependant, les règles de transformation pour la migration d'instances doivent être appropriées pour la manipulation de requêtes sur le modèle de données à notre portée, à savoir, des automates. Comme dans notre cas, les transformations sont définies entre les deux définitions de protocoles de service, nous utilisons les **expressions de chemins** pour spécifier ces règles. Ce choix est justifié par la nature des langages de requêtes associés aux automates, et d'une manière générale, utilisés dans les graphes. En effet, dans ces langages, les expressions de chemins permettent d'identifier des structures particulières dans un graphe (*chemins complet, sous-chemins, transitions, ...*), en décrivant comment atteindre ces éléments lors de navigation dans le graphe en question. Dans notre cas les graphes considérés ne sont autre que les automates représentant les deux versions du protocole du service.

Exemple 5.3 ***Transformation d'un chemin d'exécution***
A titre d'exemple, l'expression de chemin ID.VI.CC.CIC^*.VE *relative au protocole* Retraite *de la figure 5.1 exprime le chemin d'exécution de toutes les instances du service qui sont caractérisées par des traces d'exécution composées de la séquence d'activités suivante :* ID, VI *et* CC, *suivie d'un nombre de fois de l'activité* CIC, *et enfin, une dernière activité* VE.

Maintenant, considérons la règle de transformation suivante : $ID.VI.CC.CIC^*.VE \xrightarrow{Convertir} ID.VI.VE$, *entre l'automate source et un autre automate cible. Cette règle permet de* Convertir *les instances qui répondent à la requête* ID.VI.CC.CIC^*.VE *dans l'automate source, en des instances de l'automate cible qui satisfont la contrainte* ID.VI.VE. *En fait, cette règle permet de transformer toutes les traces d'exécution de la forme* ID.VI.CC.CIC^*.VE *en des traces correspondantes de la forme* ID.VI.VE, *en ignorant simplement les exécutions des activités* CC *et* CIC *dans l'automate cible. Par exemple, une instance de l'automate source ayant un chemin d'exécution* ID.VI.CC.CIC.CIC.CIC.CIC.VE *sera convertie par cette règle en une instance du nouveau automate, avec comme chemin d'exécution correspondant* ID.VI.VE.

4. Par la suite, nous utiliserons le terme anglais **mapping**

5.6.2 Formalisation du langage de migration d'instances

Le langage de migration d'instances proposé est basé, fondamentalement, sur les deux primitives suivantes :

- **les expressions générales de chemins** ;
- **les règles de transformation** ou **règles de mapping**.

A l'origine, les expressions générales de chemins (*ou simplement,* ***les expressions de chemins****, par la suite*), ont été introduites pour interroger les données semi-structurées [163]. Elles permettent d'exprimer les chemins de navigation en manipulant les **opérateurs des expressions régulières** et les **variables de chemins**. Une variable de chemin est une variable qui prend comme valeur un chemin.
Les expressions de chemins sont des primitives très utiles dans notre contexte, car elles permettent d'exprimer d'une manière uniforme, aussi bien, des **requêtes** que des **contraintes** sur les instances des protocoles [161].

La deuxième primitive se rapporte aux **règles de transformation** manipulant des expressions de chemins. Une règle de transformation utilise une expression de chemin comme une requête qui sélectionne un ensemble d'instances d'un protocole source. Dans le cas où les instances sélectionnées vérifient certaines contraintes données, alors la règle de transformation procède à leur **conversion** en des instances correspondantes dans le protocole cible. Les contraintes à satisfaire sont à leur tour exprimées comme des expressions de chemin sur le protocole cible.

Dans ce qui suit, nous introduisons les primitives du langage de migration d'une manière plus formelle.

a) Les expressions générales de chemin

Soit Σ un alphabet, et soit $\mathcal{P}$ un protocole de service. Nous désignons par Σ^* l'ensemble fini des chemins d'exécutions qui sont obtenus par la concaténation des activités de Σ en utilisant l'opérateur de concaténation ".". Ce dernier permet de joindre les noms des activités bout à bout (*par exemple :* $\mathsf{ID.VI.CC}$).

A présent, nous introduisons les notions d'expression de chemins et de variable de chemin.

On note par $\mathcal{V}$ un ensemble de variables, appelées **variables de chemins**. Une variable de chemin, associée à un protocole $\mathcal{P}$, est une variable qui prend comme valeur un chemin d'exécution dans $\mathcal{P}$.

Une évaluation sur $\mathcal{V}$ et Σ est une transformation (*mapping*) $\mu : \mathcal{V} \rightarrow \Sigma^*$ qui à chaque variable $v \in \mathcal{V}$, associe un chemin $\mu(v) \in \Sigma^*$.

Une **expression de chemin** dans un protocole est une expression régulière formée par des constantes (*i.e., des noms d'activités*), des variables de chemins et les quantificateurs classiques suivants :

- ? qui définit une expression qui existe zéro ou une fois, e.g., à l'expression de chemin $(\mathsf{a.b})^?$ correspond le chemin d'exécution vide ou le chemin d'exécution $\mathsf{a.b}$;
- $*$ qui définit une expression qui existe zéro, une ou plusieurs fois, e.g., à $(\mathsf{a.b})^*$ correspond le chemin d'exécution vide ou les chemins d'exécution $\mathsf{a.b}$, $\mathsf{a.b.a.b}$, $\mathsf{a.b.a.b.a.b}$, ...,

– + qui définit une expression qui existe une ou plusieurs fois, e.g., à $(\mathsf{a.b})^+$ correspond les chemins d'exécution : a.b ou a.b.a.b ou a.b.a.b.a.b, ...,

La notion d'évaluation est étendue aux expressions de chemins en exigeant que chaque évaluation μ doit satisfaire les deux conditions suivantes :

– μ est la fonction identité sur les éléments de Σ, i.e., $\mu(\mathsf{a}) = \mathsf{a}$, $\forall \mathsf{a} \in \Sigma$;
– pour toutes expressions de chemin x et y, nous avons $\mu(\mathsf{x.y}) = \mu(\mathsf{x}).\mu(\mathsf{y})$.

Exemple 5.4 ***variable de chemin, évaluation et expression de chemin***
A titre d'exemple, soit v *une variable de chemin et* NR *l'activité Notification Rejet du protocole Retraite de la figure 5.1.*

L'expression de chemin v.NR *correspond à tous les chemins d'exécution qui se terminent par l'activité* NR*. Ainsi,* v *peut prendre par exemple comme valeur le chemin d'exécution* ID.VI.CC.VE*. Ce chemin d'exécution est obtenu par une évaluation μ qui affecte à la variable* v *la valeur $\mu(\mathsf{v}) =$* ID.VI.CC.VE*, et par conséquent, l'évaluation de l'expression de chemins : $\mu(\mathsf{v.NR}) = \mu(\mathsf{v}).\mu(\mathsf{NR})$ donne le chemin d'exécution* ID.VI.CC.VE.NR*.*

Une autre évaluation possible $\mu(\mathsf{v})$ peut affecter la valeur ID.VI *à la variable* v*. Dans ce cas, l'évaluation $\mu(\mathsf{v.NR}) = \mu(\mathsf{v}).\mu(\mathsf{NR})$ conduit au chemin d'exécution* ID.VI.NR*.*

Par contre, il est intéressant de constater qu'il n'existe aucune expression de chemins v.EDR.CCA.NR *dans le protocole Retraite, et dont l'évaluation aboutira à un chemin d'exécution qui se termine par la séquence d'activités* EDR.CCA.NR*. En effet, aucune évaluation μ ne peut affecter à la variable* v *une valeur $\mu(\mathsf{v})$ de sorte que le chemin d'exécution résultant soit valide dans le protocole. Une telle évaluation n'est pas permise par la description du protocole, et elle est dite* ***non satisfaite****.*

On dit qu'un chemin d'exécution δ **satisfait** une expression de chemin δ_e, et on note : $\delta \models \delta_e$, s'il existe une évaluation μ, telle que $\mu(\delta_e) = \delta$.
Dans l'exemple précédent, le chemin d'exécution ID.VI.CC.VE.NR **satisfait** l'expression de chemin v.NR, contrairement au chemin d'exécution EDR.CCA.NR.

b) Utilisation des expressions de chemins

Dans notre approche, nous exploitons les expressions de chemins pour exprimer, à la fois, des requêtes et des contraintes sur les instances des protocoles de services.

Nous montrons, ci-après, comment les expressions de chemins sont utilisées par le langage de migration, aussi bien pour la sélection d'instances que pour imposer des contraintes de migration.

Soit $\mathcal{P}$ un protocole de service, et soit $Exec(\mathcal{P})$ l'ensemble de tous les chemins d'exécution possibles de $\mathcal{P}$ (*voir section 5.3*).

Dans ce qui suit, nous détaillons comment les expressions de chemins sont utilisées pour exprimer les requêtes et les contraintes sur le protocole de service $\mathcal{P}$.

• Formulation des requêtes sur les instances de protocoles
Une expression de chemin δ_e peut être utilisée pour interroger les instances d'un protocole $\mathcal{P}$. Dans ce cas, l'ensemble des réponses de δ_e contient les instances de $\mathcal{P}$ qui satisfont δ_e. Plus formellement, l'ensemble des réponses de δ_e sur $Exec(\mathcal{P})$, noté par $\delta_e(\mathcal{P})$, est défini par : $\delta_e(\mathcal{P}) = \{\delta \in Exec(\mathcal{P}) \mid \delta \models \delta_e\}$.

Exemple 5.5 ***Requête d'interrogation d'instances de protocoles***
Admettons que nous voulons filtrer toutes les instances du protocole ***Retraite*** *de la figure 5.1, qui ont atteint un état final (instances terminées), et qui ont exécuté une seule fois l'activité* CIC *(sans exécution de la boucle sur l'état Carrière Reconstituée). Une telle requête peut être formulée par l'expression de chemin, utilisant les deux variables de chemins* $\mathsf{v_1}$ *et* $\mathsf{v_2}$*, comme suit :* $\delta_e(\mathsf{Retraite}) = \mathsf{v_1.CIC.v_2}$.

L'évaluation de cette expression de chemins permet de sélectionner les différents chemins d'exécution suivants qui satisfont $\delta_e(\mathsf{Retraite})$.

- $\delta 1 = \mathsf{ID.VI.CC.CIC.VE.NR}$, *avec les évaluations :* $\mu(\mathsf{v_1}) = \mathsf{ID.VI.CC}$ *et* $\mu(\mathsf{v_2}) = \mathsf{VE.NR}$ *;*
- $\delta 2 = \mathsf{ID.VI.CC.CIC.VE.EDR.CP.CCA.PPR^*.AP}$, *avec les évaluations :* $\mu(\mathsf{v_1}) = \mathsf{ID.VI.CC}$ *et* $\mu(\mathsf{v_2}) = \mathsf{VE.EDR.CP.CCA.PPR^*.AP}$ *;*
- $\delta 3 = \mathsf{ID.VI.CC.CIC.VE.EDR.CP.CCA.PPR^*.TAD}$, *avec les évaluations :* $\mu(\mathsf{v_1}) = \mathsf{ID.VI.CC}$ *et* $\mu(\mathsf{v_2}) = \mathsf{VE.EDR.CP.CCA.PPR^*.TAD}$.

Les trois chemins d'exécution $\delta 1, \delta 2$ *et* $\delta 3$ *satisfont l'expression de chemin* $\delta_e(\mathsf{Retraite})$*, car* $\delta 1 \models \delta_e(\mathsf{Retraite}), \delta 2 \models \delta_e(\mathsf{Retraite})$ *et* $\delta 3 \models \delta_e(\mathsf{Retraite})$*, et elles sont des exécutions valides dans le protocole* ***Retraite*** *(*$\delta 1, \delta 2$ *et* $\delta 3 \in Exec(Retraite)$*).*

A chacun des trois chemins d'exécutions $\delta 1$*,* $\delta 2$ *et* $\delta 3$ *peuvent être associées différentes instances d'exécution avec des traces distinctes, en fonction du nombre d'invocations de l'activité* PPR *(*$\mathsf{PPR^*}$*).*

A partir de l'exemple précédent, nous constatons que les expressions de chemins constituent un mécanisme très puissant pour la sélection d'instances de protocoles.

• **Formulation des contraintes sur les instances de protocoles**
Une expression de chemin δ_e peut être utilisée comme une contrainte sur les instances d'un protocole $\mathcal{P}$. Nous disons qu'un chemin d'exécution $\delta \in Exec(\mathcal{P})$ satisfait une expression de chemin δ_e, et nous écrivons $\delta \models \delta_e$, si et seulement si, il existe une évaluation μ, telle que : $\mu(\delta_e) = \delta$.

Exemple 5.6 ***Expression des contraintes sur les instances***
Supposons que nous voulons vérifier si le protocole ***Retraite*** *de la figure 5.1 impose une* ***contrainte sur l'exécution*** *de l'activité Contrôle Carrière (CC) dont l'exécution doit être postérieure à l'activité Vérification Identité (VI). Une telle contrainte peut être exprimée par l'expression de chemins :* $\delta_e(\mathsf{Retraite}) = \mathsf{v_1.VI.v_2.CC.v_3}$*, avec* ($\mathsf{v_1}, \mathsf{v_2}$ *et* $\mathsf{v_3}$ *des variables de chemins). On observe que le chemin d'exécution :* $\delta = ID.VI.CC \in Exec(Retraite)$ *satisfait l'expression de chemins* $\delta_e(\mathsf{Retraite})$ *;* ($\delta \models \delta_e(Retraite)$*). Ce chemin est obtenu par l'évaluation* $\mu(\delta_e(\mathsf{Retraite})) = \mathsf{ID.VI.CC}$*, avec :* $\mathsf{v_1} = \mathsf{ID}$, $\mathsf{v_2} = \epsilon$ *et* $\mathsf{v_3} = \epsilon$*. Donc, la contrainte de postériorité d'activités est respectée par le protocole.*

c) Les règles de transformation d'instances

Soient $\mathcal{P}$ et $\mathcal{P}'$ deux protocoles de services. $\mathcal{P}$ est l'ancienne version du protocole et $\mathcal{P}'$ est sa nouvelle version, après évolution. Notre objectif principal est d'assurer la migration des instances actives dans $\mathcal{P}$ vers $\mathcal{P}'$. Nous exploitons les expressions de chemins pour spécifier des règles de transformation d'instances de la forme :

$\mathsf{r} \rightarrow \mathsf{c}$, où

r est une expression de chemins définie sur $\mathcal{P}$ et c est une expression de chemins définie sur $\mathcal{P}'$. Une telle règle prend les instances de $\mathcal{P}$ qui sont sélectionnées par la requête r et les convertit en des instances de $\mathcal{P}'$ qui satisfont l'expression de chemin c.

Par exemple, la règle a.b.c $\rightarrow$ a.e.c convertit les instances de $\mathcal{P}$ ayant le chemin d'exécution a.b.c en des instances de $\mathcal{P}'$ qui sont dans un état atteignable par le chemin d'exécution a.e.c. Cette règle simple permet de spécifier une stratégie de migration qui peut être utile dans le cas où l'activité b de $\mathcal{P}$ est replacée dans $\mathcal{P}'$ par une nouvelle activité e assurant, ainsi, une migration avec substitution d'une activité.

Il est important de noter que la possibilité d'utiliser les variables de chemins, lors de la spécification des règles de transformation, augmente l'expressivité du langage de migration proposé en le rendant **générique**.

Exemple 5.7 ***Règle de transformation d'instances***
Soit la règle de transformation : $ID.v.CIC^*.VE \rightarrow ID.VE.v$*, avec* $v \in \mathcal{V}$ *une variable de chemin. Tout d'abord, cette règle filtre toutes les instances ayant le chemin d'exécution commençant par l'activité* ID *et se terminant par une séquence arbitraire* CIC*, suivie par l'activité* VE *(e.g., le chemin* $ID.VI.CC.CIC.CIC.CIC.VE$*). Après, elle donne la possibilité de convertir les instances sélectionnées en des instances dont les traces correspondantes ne contiennent pas la séquence* CIC^**, et avec un déplacement de l'activité* VE *après* ID *(traces exprimées par le chemin* $ID.VE.VI.CC$*).*

5.7 Conclusion

Dans ce chapitre, nous avons présenté le fondement de notre approche pour la gestion des changements des protocoles des service web, et nous avons exposé son principe basé sur la formalisation d'un langage de migration d'instances.

Après la caractérisation des changements qui gouvernent l'évolution des protocoles des services web, nous avons décrit les outils conceptuels nécessaires à la spécification du langage de migration d'instances et ce dernier a été formalisé et illustré par des exemples. Basé sur les expressions de chemins et les règles de transformation, le langage proposé offre un cadre conceptuel adéquat pour la gestion de la migration des instances.

Cependant, utiliser un tel langage de bas niveau pour définir des stratégies de migration peut être lourd et encombrant pour les gestionnaires de protocoles. En effet, les règles de migration employées sont trop généralistes et ne considèrent pas les spécificités des types de changements opérés lors des évolutions des protocoles (*ajout, suppression, substitution, ...*). Par ailleurs, le langage de migration d'instances proposé est en mesure de préciser, uniquement, comment sélectionner et décrire des instances d'un ancien protocole dans le cadre du nouveau protocole. Par conséquent, plusieurs règles de transformation doivent être utilisées en même temps pour déployer une stratégie de migration particulière.

En résumé, le langage de migration d'instances s'avère limité pour l'élaboration des stratégies de migration efficaces et opérationnelles qui seront en mesure de prendre en compte des changements multiples et diversifiés en même temps. Pour surmonter cette limitation, nous présentons dans le prochain chapitre un ensemble de patterns de migration qui étendent le langage de migration générique par des abstractions de haut de niveau utiles pour les divers scénarios d'évolution récurrents.

Chapitre 6

Formalisation des patterns de migration

6.1 Introduction

Ce chapitre est consacré à l'identification et à la formalisation d'un ensemble de **patterns de migration** destinés à être utilisés pour faciliter la spécification des stratégies de migration adaptées aux besoins des utilisateurs. Les patterns de migration permettent d'étendre le langage de migration proposé dans le chapitre précédent, et de capturer différentes situations d'évolution se produisant lors des évolutions des protocoles des services web.

Le chapitre est structuré comme suit : nous commençons par décrire le concept de pattern de migration, et nous introduisons sa formalisation en section 6.2. Puis, nous abordons dans la section 6.3 les patterns de migration pour la gestion des obligations. La section 6.4 est dédiée à la présentation des patterns de migration gérant des contraintes d'interdiction. En section 6.5, la technique de composition des patterns est exposée et illustrée par des exemples réels. La section 6.6 est réservée à une étude de cas complète qui met en exergue l'utilisation concrète de notre approche pour l'élaboration de différentes stratégies de migration. Nous clôturons le chapitre par une conclusion et des discussions sur l'approche proposée, en section 6.7.

6.2 Spécification des patterns de migration

Nous commençons par rappeler la signification générale du concept de **pattern**, après nous donnons la spécification formelle et la sémantique des patterns de migration utilisés dans notre approche. A la fin de la section, nous présentons un aperçu général des patterns de migration qui sont traités dans notre étude.

6.2.1 Notion de pattern de migration d'instances

En génie logiciel, un patron de conception (**design pattern**) est un concept destiné à résoudre des problèmes récurrents. Généralement, il décrit des solutions standards pour répondre à des problèmes d'architecture et de conception des logiciels. En sa qualité de structure générique qui permet de résoudre le problème identifié, il est indépendant du langage de programmation, et il est suffisamment standardisé pour que tous puissent s'y référer [164].

Ce préambule nous ramène à la définition suivante : *Les patterns ou motifs de conception sont des recueils de bonnes pratiques de conception pour un certain nombre de problèmes récurrents* [165].

Dans le paradigme objet, les problèmes à traiter sont généralement d'ordre programmatique, entre autre, garder une forte cohésion entre les composants d'une même classe et un couplage faible entre des classes distinctes. En un mot, les patterns de conception permettent d'assurer la pérennité du code dans le temps [166].

En s'appuyant sur le concept général de pattern, nous exploitons son principe de fonctionnement pour définir des **patterns de migration** destinés à être utilisés dans le contexte de l'analyse de l'impact des changements [161]. L'objectif est d'offrir un outil conceptuel pour faciliter la spécification des stratégies de migration des instances de services. Nous donnons ci-après, une définition plus exacte d'un pattern de migration.

Définition 6.1 ***Un pattern de migration*** *capture de manière générique, dans le sens où il est paramétrable, une règle de migration fréquente. Une fois instancié, le pattern de migration donne lieu à une règle de migration, d'un ancien protocole vers un nouveau, qui cible les instances sélectionnées par le pattern.*

Les patterns proposés sont paramétrables, dans le sens où ils intègrent des variables qui peuvent être instanciées par des valeurs particulières relatives à différentes situations. Ainsi, les patterns de migration constituent un moyen très utile qui, une fois mis à la disposition des fournisseurs de services, leur donne la possibilité de définir d'**une manière déclarative les requêtes de migration d'instances**, et donc, de spécifier des stratégies de migration adaptées à leurs besoins spécifiques [161]

Lors de l'élaboration des patterns de migration, une attention particulière à été accordée à la simplicité du formalisme utilisé, mais aussi à son expressivité de manière à couvrir un large spectre de stratégies de migration possibles.

6.2.2 Formalisation des patterns de migration

Chaque pattern de migration sera décrit par les éléments suivants : son nom, sa spécification formelle et sa sémantique formelle accompagnée d'une description informelle qui explique le fonctionnement du pattern.
Soient $\mathcal{P}$ et $\mathcal{P}'$ deux protocoles décrivant des processus métiers. $\mathcal{P}$ est appelé l'ancienne version du protocole, alors que $\mathcal{P}'$ est sa nouvelle version.
Spécification des patterns de migration : En employant les expression de chemins, telles que décrites dans le chapitre précédent (*cf. ; section 5.6.2*), la spécification générale d'un pattern est la suivante :

$$(\delta_e, q_s) \xrightarrow{\text{PPM}} (\beta_e, q'_t)$$

tels que :

- δ_e est une expression de chemins sur le protocole $\mathcal{P}$ qui définit la portée du pattern ; i.e., une requête exprimée sur $\mathcal{P}$ pour sélectionner les instances actives à faire migrer par le biais de ce pattern ;
- q_s correspond à l'état de l'ancien protocole dans lequel se trouvent les instances sélectionnées par δ_e ;
- β_e est une expression de chemins qui définit le chemin d'exécution **témoin** dans le protocole $\mathcal{P}'$. Cette expression de chemin est associée aux instances qui ont été transférées à partir de l'ancien protocole $\mathcal{P}$;

- q'_t est l'état cible du nouveau protocole vers lequel les instances actives sont migrées en utilisant le chemin d'exécution témoin dérivé de δ_e ;
- PPM est un prédicat logique qui capture la sémantique du pattern. Sa spécification générale intègre, à la fois, le sous-ensemble des instances actives sélectionnées pour la migration et qui sont filtrées par l'expression de chemin δ_e, et aussi, les expressions de chemins β_e correspondantes aux traces témoins qui vérifient les contraintes de la migration. Par ailleurs, le prédicat manipule des paramètres optionnels. Le syntaxe générale du prédicat associé au pattern de migration est le suivante :

$$PPM\ ((\delta, q_s), (\beta, q'_t), param),$$

où *param* est un paramètre optionnel introduit en entrée.

Sémantique des patterns : La spécification d'un pattern de migration signifie qu'une instance active i de l'ancien protocole $\mathcal{P}$ qui a atteint un état q_s, et qui vérifie la condition δ_e, **peut migrer** vers un état q'_t du nouveau protocole $\mathcal{P}'$ qui est identifié par la contrainte β_e, si le prédicat PPM est évalué à Vrai.

Avant d'aller vers plus de détails, nous illustrons d'abord la notion de pattern par un exemple simple.

6.2.3 Exemple simple de pattern de migration

Nous illustrons, ci-dessous, la notion de pattern de migration par un exemple simple relatif au protocole Retraite de la figure 5.1. L'exploitation de ce pattern permettra d'assurer la migration d'une catégorie d'instances du protocole après son évolution.

Exemple 6.1 *Considérons le protocole Retraite représenté dans la figure 5.1, et admettons qu'une nouvelle version $\mathcal{P}'$ de ce protocole est obtenue en supprimant la boucle de l'activité Compléments Info Carrière (CIC) relative à l'état Carrière Reconstituée.*

Supposons, maintenant, que nous sommes intéressés par la migration des instances de l'ancien protocole vers des états corresponds dans le nouveau protocole $\mathcal{P}'$, mais seules les instances ayant exécuté une seule fois cette activité sont concernées par la migration. La stratégie de migration à élaborer doit transférer toutes les instances exécutées dans $\mathcal{P}'$, y compris celles qui ont déjà exécuté l'activité supprimée CIC (les instances ayant passé par la boucle une seule fois). En effet, nous désirons que l'activité CIC soit ignorée des chemins d'exécution des instances après leur migration, car le nouveau protocole ne contient pas de boucle pour cette activité. Une telle stratégie de migration pourrait être définie en utilisant le pattern de migration suivant :

$$(\delta_e, q_s) \xrightarrow{SansBoucle\ ((\delta,q_s),(\beta,q'_t),CIC)} (\beta_e, q'_t)$$

où : δ_e et β_e sont deux expressions de chemin.

La sémantique du pattern est interprétée comme suit. D'abord, la portée du pattern est donnée par toutes les instances du protocole Retraite. Chaque instance i *est caractérisée par son état courant q_s et son historique d'exécution δ qui satisfait l'expression de chemin δ_e (i.e., $\delta \in \delta_e(\mathcal{P})$). Une telle instance est* ***transférée*** *vers un état q'_t du nouveau protocole, s'il y a un chemin d'exécution β dans $\mathcal{P}'$ qui satisfait l'expression*

de chemin β_e (i.e., $\beta \in \beta_e(\mathcal{P}')$), et tel que le prédicat SansBoucle $((\delta, q_s), (\beta, q'_t), CIC)$ est évalué à Vrai. Ce prédicat est évalué à Vrai, si lorsqu'on on retire l'activité CIC de δ nous obtiendrons le chemin d'exécution β.

Par exemple, si le pattern précédent est utilisé avec les valeurs suivantes pour les expressions de chemin (avec v et v' des variables de chemins) :

- *δ_e =ID.v.VE (i.e., tous les chemins d'exécution commençant par l'activité ID et se terminant par l'activité VE), et*
- *$\beta_e = v'$ (i.e., n'importe quel chemin d'exécution dans $\mathcal{P}'$).*

Alors, les instances actives qui sont à l'état q_s (q_s=Statut Demandeur Vérifié) du protocole $\mathcal{P}$, et ayant comme chemin d'exécution actuel δ = ID.VI.CC.CIC.VE (v = VI.CC.CIC) migreront vers un état correspondant q'_t de $\mathcal{P}'$ atteignable par le chemin d'exécution β = ID.VI.CC.VE. Cette migration est due au fait que :
(i) Le chemin d'exécution ID.VI.CC.CIC.VE satisfait l'expression de chemin δ_e =ID.v.VE,
(ii) le prédicat SansBoucle $((\delta, q_s), (\beta, q'_t), CIC)$ est exprimé par :
SansBoucle $((ID.VI.CC.CIC.VE, q_s), (ID.VI.CC.VE, q'_t), CIC)$. Ce prédicat retourne la valeur Vrai, car lorsque nous enlevons l'activité CIC de δ, nous obtenons le chemin d'exécution β = ID.VI.CC.VE, et
(iii) β = ID.VI.CC.VE est un chemin d'exécution valide dans $\mathcal{P}'$ qui satisfait l'expression de chemin $\beta_e = v'$.

Le chemin d'exécution β est appelé **un chemin d'exécution témoin** pour la migration des instances actives concernées.

Avec cet exemple de pattern, un grand nombre d'instances actives de $\mathcal{P}$ peuvent migrer vers le nouveau protocole $\mathcal{P}'$, tout en supprimant les occurrences de l'activité CIC des chemins d'exécutions témoins qui seront calculés pour les instances sélectionnes.

6.2.4 Vue d'ensemble des patterns de migration

Le **Tableau ??** montre un aperçu général de l'ensemble des patterns de migration identifiés dans notre étude. Les différents patterns sont organisés conformément aux dimensions d'analyse qui ont été cernées dans la section 5.4.3. En ce sens, ils sont regroupés par type de contraintes, i.e., **obligation vs. interdiction**, et selon l'axe temporel d'analyse, i.e., **passé vs. futur**.

Pour chaque pattern identifié, nous donnons sa spécification formelle et nous expliquons sa sémantique. Nous citons, également, les cas typiques de son usage et nous l'illustrons par un exemple concret.

Dans la suite du chapitre, nous considérons que :

- δ_e et β_e sont des expressions de chemins ;
- v et v' sont des variables de chemins ;
- l'alphabet $a, b, c, d, e, f \ldots$ dénote les activités des protocoles $\mathcal{P}$ ou $\mathcal{P}'$;
- les symboles $\mathsf{i}, \mathsf{j}, \ldots$ désignent les instances actives d'un protocole de service ;
- δ : est un chemin d'exécution représentant l'historique de l'exécution d'une instance i, ayant débuté ses interactions sur la base de l'ancienne version du protocole $\mathcal{P}$. Au moment du changement de $\mathcal{P}$, l'instance i se trouve à l'état q_s ;
- β : est le chemin d'exécution témoin de l'instance i dans le nouveau protocole $\mathcal{P}'$. Ce chemin résulte de l'application d'un pattern de migration spécifique. L'état de migration associé à l'instance i dans $\mathcal{P}'$ est noté par q'_t ;

Contraintes	Aspect temporel	Pattern de migration	Label
Obligations	Exécutions passées	Compatibilité stricte	PM1
		Ré-ordonnancement d'activités	PM2
		Substitution de sous-chemin	PM3
		Réduction de protocole	PM4
		Extension de protocole	PM5
	Exécutions futures	Activités requises	PF1
		Activités obligatoires	PF2
		Substitution de sous-protocole	PF3
Interdictions	Exécutions passées	Interdiction d'activités passées	PI1
	Exécutions futures	Interdictions d'activités futures	PI2
Combinaison de patterns	Exécutions passées et/ou futures	Plusieurs possibilités et différentes stratégies	S_i

Nous débutons la présentation des patterns, par l'exposé de ceux traitant des obligations *(cf., 6.3)*, puis nous abordons les patterns imposant des interdictions *(cf.,6.4)*, et enfin nous présentons la technique de composition des patterns pour l'élaboration des stratégies de migration complexes *(cf., 6.5)*.

6.3 Patterns pour la gestion des obligations

Cette classe de patterns exprime les contraintes d'obligation à satisfaire par les instances actives, afin qu'elles puissent migrer vers la nouvelle version du protocole. Ces contraintes peuvent se rapporter aux exécutions historiques ou futures.

6.3.1 Patterns de préservation des historiques des exécutions

Les opérations de changements peuvent porter sur différentes régions du protocole initial, y compris celles dont les activités ont été déjà exécutées par des instances actives. Par conséquent, il est essentiel pour le gestionnaire de protocoles de pouvoir préciser les conditions associées aux historiques des exécutions des instances actives avant d'opérer leur migration vers le nouveau protocole.

Dans ce qui suit, nous identifions et nous analysons un ensemble de patterns de migration pour la préservation des historiques des exécutions . [1]

a. Pattern de migration pour la compatibilité stricte (PM1)

Lors de l'évolution des protocoles de service, le gestionnaire de protocole peut exiger que toutes les instances actives doivent se conformer d'une manière **stricte** aux nouvelles spécifications du protocole évolué. Ainsi, uniquement les instances actives qui satisfont les spécifications décrites par le nouveau protocole peuvent être migrées vers la nouvelle version.

1. *Les **P**atterns de **M**igration de cette classe sont notés **PMi**, où **i*** est le numéro de pattern

Le pattern assurant la migration stricte d'une instance active i, qui a réalisé un historique d'exécution δ et qui a atteint l'état q_s est exprimé, formellement, par :

$$(\delta_e, q_s) \xrightarrow{\text{Strict } ((\delta,q_s),(\beta,q'_t),-)} (\beta_e, q'_t). \quad (1)$$

Le symbole "−" dans le prédicat dénote le fait que ce pattern n'utilise aucun paramètre.

La sémantique du pattern est interprétée comme suit :
la migration d'une instance active i qui se trouve à un état $q_s \in \mathcal{P}$, et qui a parcouru un chemin d'exécution $\delta \in \delta_e(\mathcal{P})$ (*i.e., le chemin d'exécution δ de $\mathcal{P}$ satisfait l'expression de chemin δ_e*), vers un état correspondant q'_t de $\mathcal{P}'$ est possible, s'il existe un chemin d'exécution témoin β de $\mathcal{P}'$ conduisant à l'état q'_t. Le chemin d'exécution β doit, à son tour, satisfaire l'expression de chemin β_e ($\beta \in \beta_e(\mathcal{P}')$), et en même temps il doit être **identique** à δ ($\beta = \delta$). Si un tel chemin existe, alors le prédicat : *Strict* $((\delta, q_s), (\beta, q'_t), -)$ est évalué à Vrai. Comme conséquence immédiate, ce pattern permet de migrer vers la nouvelle version du protocole, les seules instances actives n'ayant pas encore atteint les régions touchées par les changements, ou celles ayant emprunté des chemins non affectés par ces changements.

Bien que très restrictif, ce pattern est utile lorsque les déviations par rapport au nouveau protocole ne sont pas autorisées. C'est le cas, par exemple, lors des changements qui expriment des évolutions critiques du protocole initial (e.g., *liées à la sécurité*), ou lorsqu'une nouvelle réglementation est introduite et exige une entrée en vigueur immédiate.

Exemple 6.2 *Supposons que le protocole Retraite de la figure 5.1 a évolué vers une nouvelle version, appelée Nretraite, telle que illustrée dans la figure 6.1, dans laquelle les modifications appliquées au protocole initial apparaissent en pointillés. Comme il peut être observé dans les figures, les changements sont mis en évidence par des zones circulaires en pointillés, et sont les suivants : la suppression des trois activités CC, VE et NR ; la suppression de l'état Vérification Terminée et l'insertion d'une nouvelle activité, Vérification Statut (**VS**). Cette nouvelle activité permet de passer de l'état Carrière Reconstituée vers l'état Statut Demandeur Vérifié, et elle assure qu'un demandeur est un salarié et non pas un propriétaire d'une profession libérale (commerçant, agriculteur, artisan...). Dans ce dernier cas, la demande est rejetée et le postulant est orienté vers la caisse de retraites des non salariés (activité NR).*

Admettons que dans le protocole évolué de la figure 6.1, la nouvelle activité de vérification VS est obligatoire et qu'elle doit être appliquée à toutes les instance actives. Pour mettre en œuvre une telle exigence qui répond à un besoin impératif, le gestionnaire de protocole peut spécifier la stratégie de migration suivante qui exploite le pattern de migration PM1 de l'équation (1) :

$$(\delta_e = v, q_s) \xrightarrow{Strict\ ((\delta,q_s),(\beta,q'_t),-)} (\beta_e = v', q'_t).$$

L'instanciation du pattern PM1 avec $\delta_e = v$ et $\beta_e = v'$ permet de convertir toute instance i de l'ancien protocole qui a un chemin d'exécution δ vers une instance du nouveau protocole, s'il existe un état q'_t dans le nouveau protocole qui soit atteignable en utilisant le chemin d'exécution δ.

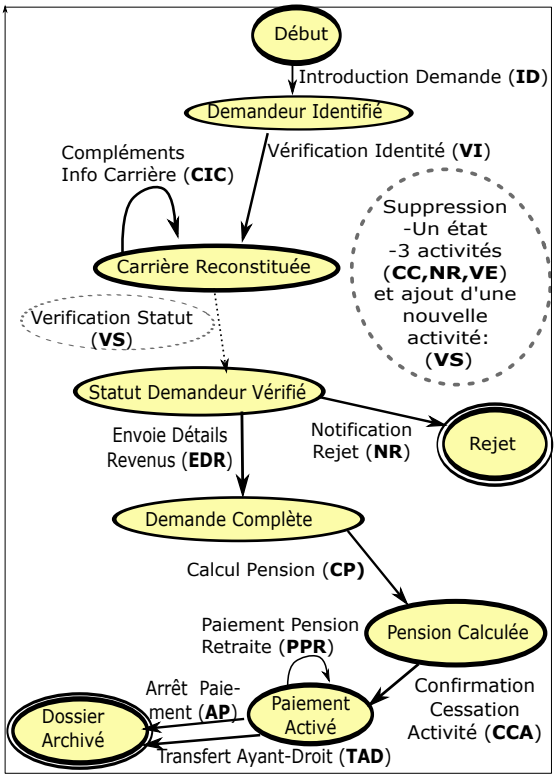

FIGURE 6.1 – Une évolution du protocole retraite (**NRetraite**).

Dans notre exemple, uniquement, les instances n'ayant pas encore atteint la région modifiée sont concernées par cette migration. Ainsi, seules les instances qui sont dans les états Demandeur Identifié ou Vérification terminée seront déplacées vers les états, respectifs, Demandeur Identifié ou Carrière Reconstituée du nouveau protocole.

Le pattern de migration stricte est, évidemment, très restrictif et n'est pas donc adapté à certaines situations. Pour pallier à cette rigidité, nous proposons dans ce qui suit, d'autres patterns de migration qui sont plus flexibles.

b. Pattern de migration pour le ré-ordonnancement d'activités (PM2)

Ce pattern exige, également, une compatibilité stricte en terme d'activités exécutées. Mais, il relâche partiellement la contrainte concernant l'ordre d'exécution des activités exigées. Sa spécification formelle est la suivante :

$$(\delta_e, q_s) \xrightarrow{\text{Reord } ((\delta,q_s),(\beta,q'_t),\alpha_e)} (\beta_e, q'_t) \tag{2}$$

Avec : α_e un paramètre du pattern qui exprime une expression de chemins.

La sémantique du pattern est donnée par le prédicat : $Reord\ ((\delta, q_s), (\beta, q'_t), \alpha_e)$ qui est évalué à Vrai pour les deux chemins d'exécutions $\delta \in \delta_e(\mathcal{P})$ et $\beta \in \beta_e(\mathcal{P}')$, si les deux conditions suivantes sont vérifiées :
(i) $\exists\ \alpha \in \alpha_e(\mathcal{P})$ et $\exists\ \delta^0, \delta^1;\ tels\ que : \delta = \delta^0\alpha\delta^1\ ;\ \beta = \delta^0\alpha'\delta^1$; et (ii) $\alpha' \in \otimes\ \alpha$.

En d'autres termes, le prédicat $Reord\ ((\delta, q_s), (\beta, q'_t), \alpha)$ renvoie Vrai, si les chemins d'exécutions δ et β sont identiques modulo un ré-ordonnancement des activités du sous-chemin α de δ.

Il convient de noter que le ré-ordonnancement autorisé par ce pattern est partiel, car il ne concerne que les activités du sous-chemin α de δ. Par conséquent, le pattern **PM2** permet de migrer des instances actives de l'ancien protocole, si les changements qui les ont affectés concernent uniquement le ré-ordonnancement de certaines activités dans leurs historiques d'exécution.

Comme pour le pattern précédent, ce pattern permet de spécifier des stratégies de migration strictes lorsque, par exemple, les changements sont critiques et doivent entrer, immédiatement, en vigueur. Il permet, toutefois, un ordre d'exécution différent pour un sous-ensemble d'activités spécifiques. Ce pattern peut être utile lorsque l'opération de changement concerne un ré-ordonnancement d'activités dans le protocole initial, alors qu'il n'est pas jugé nécessaire d'appliquer, de manière rétroactive, le nouvel ordre aux instances actives qui ont dépassé la région affectée par les changements.

Exemple 6.3 *Admettons que certains changements sont opérés sur le protocole* ***Retraite*** *de la figure 5.1. Ces changements consistent à modifier l'ordre des trois activités : VI, CC et VE. La nouvelle séquence d'exécution permise devient : VE.CC.VI, au lieu de VI.CC.VE. Par cette évolution, le gestionnaire de protocole estime qu'il est plus rentable de vérifier l'éligibilité du postulant (i.e., activité VE) avant de commencer la contrôle de la carrière (activité CC) et la vérification de son identité (activité VI), car ces deux dernières activités sont consommatrices de ressources.*

Considérons, à présent, la stratégie de migration qui utilise le pattern de migration PM2 de l'équation (2).

$$(\delta_e, q_s) \xrightarrow{Reord\ ((\delta,q_s),(\beta,q'_t),\alpha_e)} (\beta_e, q'_t)$$

avec $\delta_e = ID.v$, $\beta_e = ID.v'$ *et* $\alpha_e = VI.CC.(CIC)^*.VE$.
En utilisant cette stratégie de migration, une instance active i *de l'ancien protocole dotée d'un historique d'exécution* $\delta = ID.VI.CC.VE.EDR$ *;* ($(\mathsf{CIC})^* = \epsilon, \mathsf{v} = \mathsf{VI.CC.VE.EDR}$) *sera convertie en une instance du nouveau protocole, avec comme chemin d'exécution témoin* $\beta = ID.VE.CC.VI.EDR$ (v' *est constituée du sous-chemin* $VE.CC.VI.EDR$).

Il est important de constater que la capacité d'exprimer le paramètre α_e *du pattern PM2, par une expression de chemin, accroit sa puissance d'expressivité. En effet, dans l'exemple précédent, la séquence* α *à réordonner peut être n'importe quelle séquence qui satisfait l'expression de chemin* $\alpha_e = VI.CC.(CIC)^*.VE$. *Par exemple, pour les valeurs* $(\mathsf{CIC})^* = \mathsf{CIC.CIC.CIC}$ *et* $\mathsf{v} = \mathsf{VI.CC.CIC.CIC.CIC.VE.EDR}$, *on aura un chemin d'exécution* $\delta = ID.VI.CC.CIC.CIC.CIC.VE.EDR$. *Une instance de l'ancien protocole, avec un tel chemin d'exécution* δ, *sera convertie en une instance du nouveau protocole avec comme chemin d'exécution témoin*
$\beta = ID.VE.CC.CIC.CIC.CIC.VI.EDR$ ($\mathsf{v}' = \mathsf{VE.CC.CIC.CIC.CIC.VI.EDR}$). *Ce chemin est obtenu à partir de l'évaluation suivante du paramètre* α_e *:*
$\alpha = VI.CC.CIC.CIC.CIC.VE$ *qui est un chemin d'exécution qui satisfait l'expression de chemin* $\alpha_e = VI.CC.(CIC)^*.VE$.

c. Pattern de substitution de sous-chemins (PM3)

L'évolution d'un protocole de service peut engendrer la modification de différentes structures devenues obsolètes dans l'ancienne version, comme par exemple : la mise à jour d'une simple activité et le remplacement d'un sous-chemin ou d'un chemin complet. Cette actualisation induit la substitution de la structure concernée du protocole par une autre structure, reflétant plus fidèlement les nouveaux besoins de gestion. Dans ce contexte, le pattern de migration pour la substitution de sous-chemins permet d'établir une correspondance entre les anciens et les nouveaux sous-chemins de protocoles. Ainsi, les instances actives peuvent migrer vers le nouveau protocole, tout en remplaçant les anciens sous-chemins par de nouveaux sous-chemins à jour. La spécification formelle du pattern de substitution de sous chemins est la suivante :

$$(\delta_e, q_s) \xrightarrow{\text{Subst } ((\delta,q_s),(\beta,q'_t),(\alpha_e:\alpha'_e))} (\beta_e, q'_t) \tag{3}$$

avec : α_e et α'_e deux expressions de chemins.
Ce pattern spécifie que les instances actives ayant des historiques d'exécutions $\delta \in \delta_e(\mathcal{P})$ seront converties en des instances dans le protocole $\mathcal{P}$, avec des chemins d'exécutions témoins $\beta \in \beta_e(\mathcal{P}')$, si le prédicat $Subst\ ((\delta, q_s), (\beta, q'_t), (\alpha_e : \alpha'_e))$ est évalué à Vrai. Ce prédicat est Vrai, si et seulement si la condition suivante est vérifiée :
$\exists\ \alpha \in \alpha_e(\mathcal{P}), \exists\ \alpha' \in \alpha'_e(\mathcal{P}')$ et $\exists\ \delta^0, \delta^1\ tels\ que\ : \delta = \delta^0\alpha\delta^1\ et\ \beta = \delta^0\alpha'\delta^1$.

Donc, le prédicat $Subst\ ((\delta, q_s), (\beta, q'_t), (\alpha_e : \alpha'_e))$ renvoie Vrai, si lorsqu'on remplace dans δ, la sous-trace $\alpha \in \alpha_e(\mathcal{P})$ par $\alpha' \in \alpha'_e(\mathcal{P})$, on obtient le chemin d'exécution β.

La sémantique véhiculée par ce pattern est la suivante : une instance i ayant le chemin d'exécution $\delta \in \delta_e(\mathcal{P})$, et dont l'état courant est q_s, sera migrée vers un état q'_t du nouveau protocole $\mathcal{P}'$ qui est atteignable par le chemin d'exécution témoin $\beta \in \beta_e(\mathcal{P}')$, si le chemin d'exécution β obtenu en remplaçant dans δ la sous-trace α par α' est un chemin d'exécution valide dans $\mathcal{P}'$ et ce chemin mène à l'état $q'_t \in \mathcal{P}'$.

Ce pattern peut être utile, par exemple, lors du remplacement d'une séquence d'activités par une autre, alors que les changements n'entrent en vigueur que pour les instances futures. Dans un tel scénario, ce pattern permet de déterminer l'état cible du nouveau protocole vers lequel seront migrées les instances ayant déjà exécuté la séquence d'activités qui a subi des modifications. L'application de ce pattern de migration assure la mise en conformité des instances actives avec la nouvelle spécification du protocole, mais sans effets rétro-actifs.

Exemple 6.4 *Considérons de nouveau le protocole* ***Retraite*** *de la figure 5.1, et supposons que l'organisme de gestion des retraites a effectué des changements dans sa logique métier pour améliorer ses services. Le nouveau protocole est illustré dans la figure 6.2, où les changements sont mis en évidence par un ovale en pointillées.*

*Une première amélioration du protocole consiste à permettre aux retraités de choisir leur mode de paiement (par chèque, par mandat carte, paiement à domicile, . . .). Cette possibilité est réalisée par la nouvelle activité : Sélectionner Mode Paiement (***SMP***). Une autre modification du protocole consiste à opérer des retenues pour les mutuelles et compagnies tierces ; activité : Choisir Mutuelles (***CM***). La dernière actualisation du protocole donne la possibilité d'extraire certaines informations relatives aux demandeurs, à partir des systèmes d'informations d'autres partenaires.*

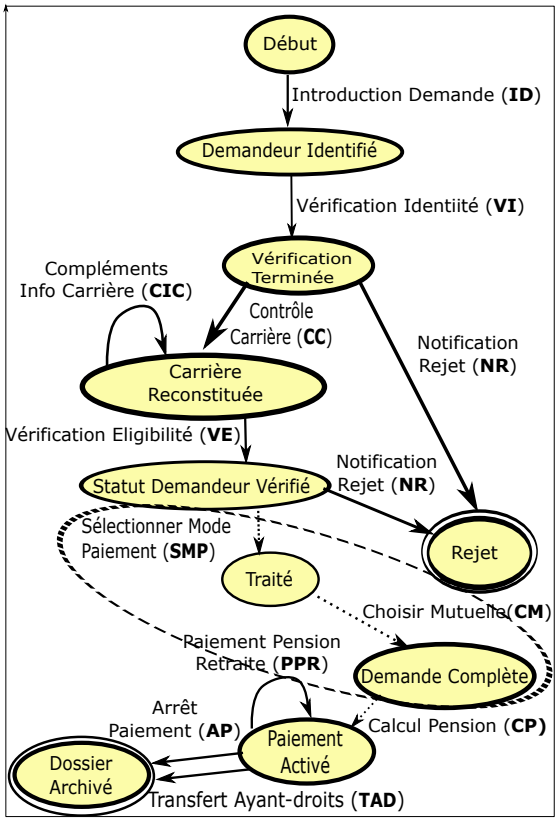

FIGURE 6.2 – Substitution d'un sous-chemin du protocole **Retraite**.

Concernant cette dernière modification, les anciennes activités Envoie Détails Revenus (**EDR**) *et* Confirmation Cessation Activité (**CCA**) *sont effectuées d'une manière transparente et sont intégrées à d'autres activités, par exemple,* Vérification Éligibilité **(VE)**. *Par conséquent, les deux anciennes activités* ***(EDR)*** *et* ***(CCA)*** *sont supprimées du nouveau protocole. Les autres chemins du protocole restent inchangés.*

Lors de la migration vers le nouveau protocole, les instances actives ayant emprunté des chemins concernés par les changements ne peuvent continuer leurs exécutions, sauf si les contraintes décrites dans l'équation ***(3)*** *sont vérifiées. Considérons, maintenant, la stratégie de migration suivante qui est basée sur le pattern* ***PM3*** *de l'équation (3).*

$$(\delta_e, q_s) \xrightarrow{Subst\ ((\delta,q_s),(\beta,q'_t),(\alpha_e:\alpha'_e))} (\beta_e, q'_t)$$

avec $\delta_e = v$, $\beta = v'$, $\alpha_e = EDR.CP.CCA$ *et* $\alpha'_e = SMP.CM.CP$.

Avec cette stratégie de migration, les instances de l'ancien protocole qui ont atteint l'état Paiement Activé *en parcourant, par exemple, le chemin d'exécution suivant :* $\delta = ID.VI.CC.CIC.VE.EDR.CP.CCA.PPR$ *seront converties en des instances du nouveau protocole de la figure 6.2 qui sont au même état* Paiement Activé, *mais avec le chemin d'exécution témoin* β *suivant :* $\beta = ID.VI.CC.CIC.VE.SMP.CM.CP.PPR$.

Le chemin β *est obtenu suite au remplacement du sous-chemin* $\alpha = EDR.CP.CCA$ *(qui satisfait l'expression de chemin* $\alpha_e(P)$*) par le sous-chemin* $\alpha' = SMP.CM.CP$ *(qui satisfait* $\alpha'_e(P')$*), et* β *est un chemin d'exécution valide dans le nouveau protocole.*

d. Pattern de migration lors de la réduction d'un protocole (PM4)

Ce pattern d'obligation permet d'ignorer certaines activités qui ont été déjà exécutées dans le protocole initial. Sa spécification formelle est la suivante :

$$(\delta_e, q_s) \xrightarrow{\text{Reduc } ((\delta,q_s),(\beta,q'_t),\alpha_e)} (\beta_e, q'_t) \tag{4}$$

où α_e est une expression de chemin.
La sémantique du pattern est exprimée par le prédicat $Reduc\ ((\delta, q_s), (\beta, q'_t), \alpha_e)$ qui est évalué à Vrai, si et seulement si : $\exists\ \alpha \in \alpha_e(\mathcal{P})$ et $\exists\ \delta^0, \delta^1$, tels que : $\delta = \delta^0\alpha\delta^1$ et $\beta = \delta^0\delta^1$. Donc, le prédicat $Reduc\ ((\delta, q_s), (\beta, q'_t), \alpha_e)$ renvoie Vrai, si le chemin d'exécution β est obtenu en supprimant la séquence d'activités α de δ.

La stratégie de migration supportée par ce pattern est interprétée comme suit : Le pattern **PM4** permet de faire migrer une instance active i qui a un historique d'exécution $\delta \in \delta_e(\mathcal{P})$, et dont l'état courant dans l'ancien protocole est q_s, vers un état q'_t du nouveau protocole, si et seulement si l'état q'_t est atteignable par le chemin d'exécution témoin $\beta \in \beta_e(\mathcal{P}')$ qui est obtenu en supprimant la séquence α de δ.

Ce pattern est utilisé pour assurer la migration des instances actives vers un nouveau protocole, suite à des opérations de suppression d'une ou de plusieurs activités consécutives du protocole initial.

Exemple 6.5 *Pour illustrer l'exploitation du pattern de migration associé à la réduction du protocole, considérons à nouveau le protocole initial Retraite de la figure 5.1. Pour répondre à des besoins d'évolution (on ne s'intéresse pas aux raisons de ces changements), le gestionnaire de protocole a décidé de supprimer les deux activités : CIC et VE. Dans le protocole résultant de l'application de ces changements, une instance* i *sera capable de transiter directement de l'état Vérification Terminée à l'état Statut Demandeur Vérifié quand l'activité CC est invoquée.*

L'instanciation du pattern PM4 avec les paramètres $\delta_e = v$, $\beta_e = v'$ et $\alpha_e = CIC^*.VE$ donne :

$$(v, q_s) \xrightarrow{Reduc\ ((\delta,q_s),(\beta,q'_t),CIC^*.VE)} (v', q'_t)$$

Cette stratégie de migration permettra de transférer toute instance i*, dont l'historique d'exécution dans l'ancien protocole est δ, vers un état q'_t du nouveau protocole qui sera atteignable par un chemin d'exécution β. Ce dernier est obtenu à partir de δ, en supprimant le sous-chemin qui satisfait l'expression de chemin $\alpha_e = CIC^*.VE$. Par exemple, une instance* i *de l'ancien protocole qui est dotée d'un historique $\delta_1 = ID.VI.CC.CIC.VE.EDR$ sera sélectionnée par cette stratégie, car $\delta_1 \in \delta_e(Retraite)$), et elle sera déplacée vers un état du nouveau protocole qui est atteignable par le chemin d'exécution témoin $\beta_1 = ID.VI.CC.EDR$ (β_1 est obtenu après la suppression de la séquence $CIC.VE$ de δ_1).*

Ce pattern pourra, également, être appliqué pour les instances ayant comme historiques d'exécutions :
$\delta_2 = ID.VI.CC.VE.EDR$ et $\delta_3 = ID.VI.CC.CIC.CIC.CIC.VE.EDR$ qui seront converties au même chemin d'exécution témoin $ID.VI.CC.EDR$. Ce chemin est obtenu en supprimant VE et $CIC.CIC.CIC.VE$ de δ_2 et δ_3, respectivement.
(noter que : $\beta 1 = \beta 2 = \beta 3 = ID.VI.CC.EDR$ est un chemin d'exécution valide dans le nouveau protocole).

e. Pattern de migration lors de l'extension d'un protocole (PM5)

Ce pattern est similaire au pattern précédent, mais fonctionne d'une manière opposée dans le sens où il prend en charge l'extension du protocole par une nouvelle séquence d'activités. Il permet, dans ce cas, aux instances qui ont déjà exécuté la région modifiée de migrer vers les états correspondants dans le nouveau protocole, tout en ignorant les activités ajoutées. La spécification de ce pattern est la suivante :

$$(\delta_e, q_s) \xrightarrow{\text{Extend } ((\delta,q_s),(\beta,q'_t),\alpha_e)} (\beta_e, q'_t) \tag{5}$$

où α_e est une expression de chemin.
Le prédicat $Extend\ ((\delta, q_s), (\beta, q'_t), \alpha_e)$ est évalué à Vrai ssi : $\exists\ \alpha \in \alpha_e(\mathcal{P})$ et $\exists\ \delta^0, \delta^1$ tels que : $\delta = \delta^0\delta^1$ *et* $\beta = \delta^0\alpha\delta^1$.

La sémantique véhiculée par le pattern **PM5** est interprétée comme suit : une instance particulière i, ayant un historique d'exécution $\delta \in \delta_e(\mathcal{P})$ et un état courant q_s, peut migrer vers un état correspondant q'_t dans le nouveau protocole $\mathcal{P}'$, s'il existe un chemin d'exécution témoin $\beta \in \beta_e(\mathcal{P}')$ qui est obtenu en insérant la séquence α dans le chemin d'exécution δ. Ce pattern est, généralement, utile lors des évolutions de protocoles induisant des ajouts de séquences d'activités dans le protocole initial.

Exemple 6.6 *Considérons encore le protocole* ***Retraite*** *de la figure 5.1 et admettons que les nouvelles lois de la retraite stipulent qu'après la reconstruction de la carrière du demandeur, l'organisme de gestion des retraites doit envoyer aux postulants une notification préalable indiquant le total des trimestres validés à prendre en compte pour le calcul de la pension. Cette information doit être soumise à l'accord préalable du demandeur. L'évolution du protocole retraite conformément à cette nouvelle réglementation est représentée dan la figure 6.3.*

*Comme il apparait dans le nouveau protocole (zone en pointillés), une nouvelle activité Notification Préalable (***NP***) est insérée à partir de l'état Carrière Reconstituée vers un nouveau état Notifié. A partir de ce nouveau état, le demandeur a la possibilité, soit de valider la notification reçue, soit de transmettre ses réclamations en cas de désaccord avec le contenu. Dans les deux cas, une nouvelle activité Réponse* **(R)** *doit être exécutée par le postulant. Dés l'exécution de cette activité R, l'instance du service transite vers un nouveau état Confirmé et peut continuer son exécution sans d'autres changements dans le protocole (la prochaine activité à exécuter est VE).*

Considérons une stratégie de migration basée sur le pattern ***PM5*** *qui utilise les paramètres suivants :* $\delta_e = v$, $\beta_e = v'$ *et* $\alpha_e = NP.R$. *Une telle stratégie permet de convertir les instances de l'ancien protocole en des instances du nouveau protocole, tout en insérant les nouvelles activités. Par exemple, une instance active* i *avec un historique d'exécution* $\delta 1 = ID.VI.CC$ *sera migrée vers l'état Carrière Reconstituée du nouveau protocole avec comme chemin d'exécution témoin* $\beta_1 = \delta_1$ *(cette instance n'a pas encore atteint la région modifiée du protocole). Soit, à présent, une autre instance* j *qui est à l'état Demande Complète de l'ancien protocole avec un historique d'exécution* $\delta 2 = ID.VI.CC.VE.EDR$. *Bien que l'instance* j *est non conforme avec le nouveau protocole, notre stratégie de migration basée sur le paramétrage du pattern PM5 permettra de faire migrer l'instance* j *vers l'état Demande complète du nouveau protocole, avec comme chemin d'exécution témoin* $\beta 2 = ID.VI.CC.NP.R.VE.EDR$. *Le chemin d'exécution* $\beta 2$ *est obtenu suite à l'insertion de la séquence* $NP.R$ *dans* $\delta 2$.

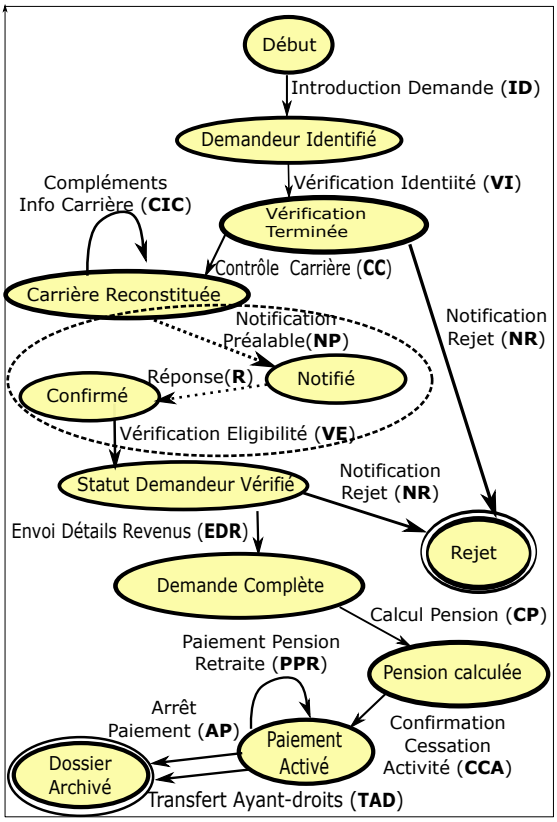

FIGURE 6.3 – Une évolutin du protocole **Retraite** exprimant l'extension.

6.3.2 Patterns assurant des obligations futures

Dans les analyses précédentes, uniquement les exécutions passées (*les historiques*) ont été considérées pour définir des stratégies de migration des instances actives vers un nouveau protocole. Nous complétons notre analyse relative aux contraintes d'obligations, par la proposition d'autres patterns additionnels qui se concentrent sur la spécification des contraintes sur les exécutions futures des instances. [2]

a. Pattern de migration d'activités requises (PF1)

Ce pattern impose l'existence d'une séquence d'activités requises, notée λ, parmi les chemins d'exécution futurs du nouveau protocole. Cette contrainte d'obligation est garantie, en exigeant qu'il existe, au moins, un chemin d'exécution futur qui contient la séquence d'activité λ.

La spécification formelle du pattern **Requise** est la suivante :

$$(\delta_e, q_s) \xrightarrow{\text{Requise } ((\delta,q_s),(\beta,q'_t),\lambda_e)} (\beta_e, q'_t) \tag{6}$$

où λ_e est une expression de chemins.
Soit $\mathcal{C}_{\mathcal{SP}'}(P', q'_t)$ le sous protocole de $\mathcal{P}'$, dont l'état initial est l'état q'_t.

2. *Les **P**atterns de migration assurant des obligations **F**uture sont notés **PFi**, avec **i** le numéro du pattern.*

Le prédicat *Requise* $((\delta, q_s), (\beta, q'_t), \lambda_e)$ est évalué à Vrai si : (i) $\exists \lambda \in \lambda_e(\mathcal{P}')$; et (ii) $\exists w \in L(\mathcal{C}_{\mathcal{SP}'}(\mathcal{P}', q'_t))$, tels que $w = w^0.\lambda.w^1$ (w^0 *et* w^1 *sont des sous-chemins d'exécution de* $\mathcal{C}_{\mathcal{SP}'}(\mathcal{P}', q'_t)$*). Autrement dit, il existe un chemin d'exécution* w *qui contient la séquence* λ. Si un tel chemin existe, alors le prédicat :*Requise* $((\delta, q_s), (\beta, q'_t), \lambda_e)$ associé au pattern **Requise** est évalué à Vrai et la migration est satisfaite.

La sémantique de ce pattern est la suivante : une instance i, ayant un chemin d'exécution $\delta \in \delta_e(\mathcal{P})$ et un état courant q_s, sera migrée vers un état correspondant q'_t dans le nouveau protocole $\mathcal{P}'$ avec comme chemin d'exécution témoin $\beta \in \beta_e(\mathcal{P}')$, si l'instance i disposera de la possibilité d'exécuter la séquence $\lambda \in \lambda_e(\mathcal{P}')$ parmi les exécutions futures qui lui sont offertes à partir de q'_t. C'est à dire, il existe au moins un chemin d'exécution w qui contient la séquence λ, et qui mène de l'état q'_t vers un état final du nouveau protocole $\mathcal{P}'$.

Ce pattern permet, ainsi, de s'assurer que certaines activités critiques (e.g., *paiement, annulation, validation*, ...) sont toujours disponibles pour les instances après leur migration. Le pattern d'activités requises est utile, par exemple, lorsque des activités ont été déplacées et que l'on souhaite s'assurer que ces activités seront disponibles pour certaines instances après leur migration vers le nouveau protocole. Il permet aussi de s'assurer, que pour certaines instances, la migration vers le nouveau protocole respecte les engagements initiaux pris vis à vis des clients (e.g., *la possibilité pour un client d'annuler l'exécution en cours*).

Exemple 6.7 *A titre d'illustration, supposons que les clients du service web* ***Retraite*** *ont commencé leur interactions conformément au protocole de service décrit dans la figure 5.1, et admettons que certaines instances actives ont atteint l'état* Carrière Reconstituée*. Lors de l'évolution du protocole de service, les changements opérés consistent à supprimer l'activité :* Complément Info Carrière **(CIC)***. Néanmoins, les instances actives ont commencé leurs interactions sous l'hypothèse que l'activité :* $\lambda \in \lambda_e(\mathcal{P}')$=**(CIC)** *fait partie des activités du protocole, et qu'elle peut être invoquée d'une manière itérative (boucle sur l'activité* (CIC^*)*). C'est la raison pour laquelle certains postulants, ayant introduit leur carrière progressivement par l'exécution répétitive de cette activité, n'ont pas encore terminé cette tâche. Or, dans le nouveau protocole, cette possibilité n'est plus disponible. Pour assurer la migration des instances actives vers le nouveau protocole, tout en garantissant des interactions futures correctes, l'activité* ***(CIC)*** *doit apparaitre dans les chemins d'exécutions futurs, autrement les conversations seront compromises et conduiront à des incompatibilités entre le contrat du client et celui du fournisseur.*

L'application du pattern de migration ***PF1*** *de l'équation (6), avec comme paramètre l'activité requise* λ= CIC*, garantit la migration des instances actives, si cette activité figure au moins une fois dans les sous-chemins d'exécution futurs du nouveau protocole (appelé* RetraiteNouveau*). Cette condition est concrétisée en imposant qu'il existe un chemin d'exécution complet dans le sous-protocole :*
$\mathcal{C}_{\mathcal{SP}'}$(RetraiteNouveau, Carrière Reconstituée)*, qui contient l'activité requise* CIC*. La satisfaction de cette condition attribue une valeur* Vrai *au prédicat :* Requise *((*δ*,* Carrière Reconstituée*), (*β*,* Carrière Reconstituée*),* CIC*), ce qui conduit à une migration correcte des instances actives concernées.*

b. Pattern de migration pour des activités obligatoires (PF2)

L'objectif de ce pattern est de forcer les instances à exécuter une séquence d'activités (*ou une simple activité*), après leur migration vers le nouveau protocole. La séquence d'activités en question est dite **obligatoire** et la spécification du pattern est la suivante :

$$(\delta_e, q_s) \xrightarrow{\text{Obligatoire } ((\delta,q_s),(\beta,q'_t),\lambda_e)} (\beta_e, q'_t) \tag{7}$$

où λ_e est une expression de chemins.

Soit $\mathcal{C}_{\mathcal{SP}'}(P', q'_t)$, le sous-protocole de $\mathcal{P}'$, dont l'état initial q'_t est celui correspondant à la migration des instances actives. Le prédicat : *Obligatoire* $((\delta, q_s), (\beta, q'_t), \lambda_e)$ est évalué à Vrai si : $\forall\, w \in L(\mathcal{C}_{\mathcal{SP}'}(\mathcal{P}', q'_t)), \exists \lambda \in \lambda_e(\mathcal{P}')$, tel que : $w = w^0 \lambda w^1$ (w^0 et w^1 sont des sous-chemins de $\mathcal{C}_{\mathcal{SP}'}(\mathcal{P}', q'_t)$). Autrement dit, tous les chemins d'exécution w de $\mathcal{C}_{\mathcal{SP}'}(P', q'_t)$ doivent contenir la séquence obligatoire λ.

La sémantique du pattern **PF2** est la suivante : une instance i, ayant un historique d'exécution $\delta \in \delta_e(\mathcal{P})$ et un état courant q_s, sera transférée vers un état correspondant q'_t dans le nouveau protocole $\mathcal{P}'$ avec comme chemin d'exécution témoin $\beta \in \beta_e(\mathcal{P}')$, si cette instance n'a pas d'autres choix possibles que d'exécuter la séquence $\lambda \in \lambda_e(\mathcal{P}')$, avant de terminer son exécution dans le nouveau protocole. Dans ce cas, tous les chemins qui mènent de l'état q'_t vers un état final du nouveau protocole $\mathcal{P}'$ contiennent la séquence d'activités obligatoires λ.

Contrairement au pattern précédent **PF1** qui donne la possibilité aux instances migrées de choisir un chemin futur contenant la séquence requise d'activités, le pattern **PF2** impose la présence de la séquence d'activités obligatoires λ dans tous les chemins d'exécution futurs. Ce pattern est utile, par exemple, pour imposer l'exécution d'une activité critique (e.g., identification, paiement, contrôle de sécurité, ...) à des instances qui ont été transférées vers un nouveau protocole.

c. Pattern de migration pour la substitution de sous protocole (PF3)

Les deux patterns précédents se focalisent, essentiellement, sur des activités simples ou sur des séquences d'activités. Or, dans la réalité, il est plus intéressant de considérer tout un sous-protocole contenant plusieurs sous-chemins et différentes structures de branchement. Ces structures expriment différents points de décision au niveau du protocole de service. Du point de vue fonctionnel, un sous-protocole décrit des procédures de gestion particulières qui sont associées à un processus métier spécifique.

La spécification formelle du pattern de migration pour la substitution d'un sous protocole est la suivante :

$$(\delta_e, q_s) \xrightarrow{\text{Sous-P } ((\delta,q_s),(\beta,q'_t),(S_P:S_{P'}))} (\beta_e, q'_t) \tag{8}$$

avec comme paramètres : $\mathcal{S}_{\mathcal{P}} = \mathcal{C}_{\mathcal{SP}}(P, q_s)$ qui représente le sous-protocole de $\mathcal{P}$ ayant l'état initial q_s, et $\mathcal{S}_{\mathcal{P}'} = \mathcal{C}_{\mathcal{SP}'}(P', q'_t)$ le sous-protocole de $\mathcal{P}'$ dont l'état initial est q'_t.

Le prédicat *Sous-P* $((\delta, q_s), (\beta, q'_t), (\mathcal{S}_{\mathcal{P}} : \mathcal{S}_{\mathcal{P}'}))$ est évalué à Vrai, si l'ancien sous-protocole $\mathcal{S}_{\mathcal{P}}$ est simulé par le nouveau sous-protocole $\mathcal{S}_{\mathcal{P}'}$., c'est à dire :

$$\mathcal{C}_{\mathcal{SP}}(P, q_s) \ll \mathcal{C}_{\mathcal{SP}'}(P', q'_t).$$

La sémantique du pattern **PF3** est interprétée comme suit : une instance active i, ayant un chemin d'exécution $\delta \in \delta_e(\mathcal{P})$ et un état courant q_s, sera transférée vers un état correspondant q'_t dans le nouveau protocole $\mathcal{P}'$ avec comme chemin d'exécution témoin $\beta \in \beta_e(\mathcal{P}')$, si en commençant à partir de q'_t, l'instance i peut exécuter n'importe quelle séquence d'opérations qui était possible à partir de q_s. La relation de simulation $\mathcal{C}_{\mathcal{SP}}(P, q_s) \ll \mathcal{C}_{\mathcal{SP}'}(P', q'_t)$ permet de s'assurer que l'état q'_t, cible de la migration, peut reproduire tous les comportements qui étaient possibles à partir de l'ancien état q_s.

Ce pattern est utile, par exemple, pour assurer la migration des instances actives qui sont à l'état q_s de l'ancien protocole, vers un état q'_t du nouveau protocole, et cette migration est **transparente** pour les clients (e.g., *respect des engagements envers les clients, car tout ce qui était possible dans l'ancien protocole reste, également, possible dans le nouveau protocole*). Ainsi, ce pattern assure des garanties de préservation des anciennes règles métier (*exprimées par* $\mathcal{C}_{\mathcal{SP}}(P, q_s) \in \mathcal{P}$), et qui sont incluses dans le nouveau sous-protocole $\mathcal{C}_{\mathcal{SP}'}(P', q'_t) \in \mathcal{P}'$. En ce sens, la mise en œuvre de ce pattern assure le respect des anciens contrats de services, lors des exécutions futures.

*(L'utilisation du pattern de migration **PF3** est illustrée en détails dans la section étude de cas 6.6)*

6.4 Patterns pour la gestion des interdictions

Un aspect complémentaire aux contraintes d'obligation présentées dans la section précédente concerne la prise en compte des interdictions. [3]

Le but de la prise en compte des contraintes d'interdiction est de renforcer les conditions de migration, en spécifiant des garanties supplémentaires pour assurer une migration cohérente. Une contrainte d'interdiction peut porter sur les historiques des instances, aussi bien que sur les exécutions futures.

Nous présentons, dans ce qui suit, deux patterns d'interdiction.

a. Interdiction d'une séquence d'activités dans le passé (PI1) :

Ce pattern est destiné à restreindre la migration aux seules instances actives n'ayant pas encore exécuté une séquence particulière d'activités durant leurs interactions passées.

La spécification formelle du pattern de **prohibition passée** est la suivante :

$$(\delta_e, q_s) \xrightarrow{\text{Interdit-P } ((\delta,q_s),(\beta,q'_t),\gamma_e)} (\beta_e, q'_t) \tag{9}$$

où γ_e est une expression de chemins.
Le prédicat Interdit-P $((\delta,q_s),(\beta,q'_t), \gamma_e)$ est évalué à Vrai si et seulement si :$\nexists\ \gamma \in \gamma_e(\mathcal{P})$ et $\nexists\ \delta^0, \delta^1$, tels que : $\delta = \delta^0\gamma\delta^1$, ($\delta^0$ *et* δ^1 *sont deux séquences d'activités*). En d'autres termes, le prédicat renvoie la valeur Vrai, si le chemin d'exécution historique δ ne contient pas la séquence $\gamma \in \gamma_e(\mathcal{P})$. C'est à dire que l'activité interdite n'a pas été invoquée dans le passé de l'instance à faire migrer.

3. *Les **P**atterns de migration pour la gestion des contraintes d'**I**nterdictions sont notés **PIi**, où **i*** est le numéro du pattern.

La sémantique du pattern est la suivante : la migration d'une instance i ayant atteint l'état q_s et dont le chemin d'exécution est $\delta \in \delta_e(\mathcal{P})$, vers un état correspondant q'_t du nouveau protocole, avec comme trace témoin $\beta \in \beta_e(\mathcal{P}')$, est possible si l'instance i n'a pas déjà exécuté la séquence d'activité γ.

Ce pattern est utile si la progression future des procédures déjà déclenchées dépend de ce qui a été réalisé dans le passé.

Exemple 6.8 *Dans cet exemple, inspiré du domaine du commerce électronique, nous considérons qu'une activité* Accorder Remise *a été ajoutée à la nouvelle spécification d'un protocole de* Gestion des commandes clients *qui a subi des évolutions. Cette nouvelle activité permet d'accorder des remises pour une certaine catégorie de commandes clients. Conformément aux nouvelles règles métiers, la remise n'est accordée que pour les clients n'ayant pas déjà confirmé la livraison (activité* Confirmer Livraison*). Ainsi, la migration des instances actives vers le nouveau protocole n'est possible que si l'activité* Confirmer Livraison *n'a pas été exécutée dans l'historique des exécutions. L'application du pattern de migration* ***PI1*** *de l'équation (9) assure une migration cohérente vers le nouveau protocole pour les seules instances actives dont les historiques d'exécution ne contiennent pas l'activité* γ=Confirmer Livraison.

b. Pattern d'interdiction d'activités future (PI2) :

Après la migration vers le nouveau protocole d'une instance active i, ce pattern interdit l'exécution par i d'une séquence spécifique d'activités futures.

La formalisation du pattern **prohibition future** est la suivante :

$$(\delta_e, q_s) \xrightarrow{\text{Interdit-F } ((\delta,q_s),(\beta,q'_t),\gamma_e)} (\beta_e, q'_t) \tag{10}$$

où γ_e est un expression de chemins.
Soit $\mathcal{C}_{\mathcal{SP}'}(\mathcal{P}', q'_t)$ le sous-protocole de $\mathcal{P}'$ ayant comme état initial q'_t.

Le prédicat : *Interdit-F* $((\delta, q_s), (\beta, q'_t), \gamma_e)$ est évalué à Vrai si : $\forall\, w \in L(\mathcal{C}_{\mathcal{SP}'}(\mathcal{P}', q'_t))$, $\nexists \gamma \in \gamma_e(\mathcal{P}')$, tels que : γ est un sous-chemin de w. Plus simplement, il n'existe aucune exécution possible dans le nouveau protocole qui contient la séquence γ, et qui mènent de l'état q'_t de la migration à un état final du protocole.

La sémantique de ce pattern est la suivante : la migration d'une instance i ayant atteint l'état q_s, et dont l'historique d'exécution est $\delta \in \delta_e(\mathcal{P})$, vers un état correspondant q'_t du nouveau protocole est possible, si et seulement si l'instance i n'a pas la possibilité d'exécuter la séquence d'activités λ à partir de son nouvel état de migration q'_t. En d'autres termes, l'instance i peut migrer, si la séquence $\gamma \in \gamma_e(\mathcal{P}')$ ne figure dans aucun chemin d'exécution du sous-protocole $\mathcal{C}_{\mathcal{SP}'}(\mathcal{P}', q'_t)$.

Ce pattern peut, par exemple, être utile pour éviter à certaines instances de faire des annulations de leurs exécutions, après la migration vers le nouveau protocole. A titre d'exemple, dans la majorité des protocoles certaines activités critiques, telles que : annuler, défaire, quitter,... sont interdites après la migration. Le pattern peut servir, également, pour garantir l'application de certaines règles métiers particulières, par la restriction des choix offerts aux instances actives ayant démarré conformément à l'ancien protocole.

6.5 Composition des patterns de migration

Les patterns présentés précédemment sont basiques et ne permettent de prendre en charge, qu'une seule propriété à respecter lors de la migration. De ce fait, ils ne permettent de réaliser que des stratégies de migration simples. Cependant, dans les processus métiers réels, les stratégies de migration sont plus complexes et peuvent intégrer plusieurs contraintes en même temps. Ainsi, des patterns plus élaborés et plus riches sont nécessaires, afin de pouvoir gérer plusieurs types de contraintes dans le cadre d'une même stratégie de migration.

Pour remédier à cette limitation, nous proposons une technique de **composition** des patterns de migration de base. Cette technique est fondée sur la combinaison des patterns précédents, de façon à pouvoir traiter plusieurs contraintes à la fois. Plus concrètement, la composition est réalisée en articulant les patterns de base par les connecteurs logiques classiques (*conjonction, disjonction, ...*), et elle permet de mettre en œuvre des stratégies de migration plus sophistiquées [161].

Pour montrer l'importance de cet aspect, trois cas de composition de patterns sont exposés ci-après.

6.5.1 Pattern imposant plusieurs obligations passées

La composition du pattern de migration **PM5** avec le pattern **PM3**, en utilisant l'opérateur de conjonction, peut être définie comme suit :

$$(\delta_e, q_s) \xrightarrow{\text{Extend } ((\delta,q_s),(\beta,q'_t),\lambda_e) \wedge \text{Subst } ((\delta,q_s),(\beta,q'_t),(\alpha_e:\alpha'_e))} (\beta_e, q'_t) \tag{11}$$

où α_e, α'_e et λ_e sont des expressions de chemins.

Cette stratégie de migration, nommée S1, est obtenue par la conjonction des patterns **PM5** et **PM3**. Elle intègre, à la fois, la sémantique du pattern **PM5** et celle du pattern **PM3**. Ainsi, la spécification précédente définie une nouvelle stratégie qui permet de migrer les instances actives qui vérifient, en même temps, le pattern **PM5** et le pattern **PM3**.

Exemple 6.9 *Considérons le protocole* ***Retraite*** *de la figure 5.1 et sa nouvelle version* ***NRetraite*** *montrée dans la figure 6.1. Comme, il a été expliqué précédemment, les activités CC, NR et VE ont été supprimées. Il en est de même pour l'état Vérification Terminée. En plus, la nouvelle activité VS a été ajoutée (les changements sont mises en évidence dans la figure 6.1 avec des pointillés).*

Considérons maintenant une instanciation de la spécification de l'équation (11) avec les paramètres suivants : $\delta_e = v.NR$, $\beta_e = v'$, $\alpha_e = CC.v.VE$, $\alpha'_e = v.VS$ et $\lambda_e = VS$. Cette instantiation définie une nouvelle stratégie de migration, dont la sémantique est décrite ci-après.

La portée de la stratégie S1 est définie par l'expression de chemins $\delta_e = v.NR$ et inclue toutes les instances de l'ancien protocole ayant des historiques d'exécution qui se terminent par l'activité NR (i.e., les éléments de $\delta_e(Retraite)$, avec $\delta_e = v.NR$). Conformément à la figure 5.1, toutes ces instances sont à l'état Rejet de l'ancien protocole, mais avec différents historiques possibles.

Comme il apparait dans la spécification de la stratégie, l'expression de chemin $\beta_e = v'$ n'exige pas de contraintes particulières sur le chemin d'exécution témoin. La stratégie S1 pourra, alors, migrer les instances qui sont à l'état Rejet de l'ancien protocole vers l'état Rejet du nouveau protocole ***NRetraite****, mais doit générer différents chemins d'exécution témoins, en fonction des historiques des instances. Le pattern d'extension PM5 sera appliqué aux instances de l'ancien protocole qui ont exécuté l'activité NR à partir de l'état Vérification Terminée. Ces instances ont un historique d'exécution $\delta 1 = ID.VI.NR$, et elles auront comme chemin d'exécution témoin $\beta 1 = ID.VI.VS.NR$. Ce chemin témoin est obtenu par l'extension de $\delta 1$ avec l'activité VS. A signaler que, VS est intuitivement conforme avec l'expression $\alpha'_e = v.VS$ du pattern. En plus, le chemin d'exécution résultant $\beta 1$ est un chemin d'exécution du nouveau protocole qui conduit à l'état Rejet.*

Par contre, les instances à l'état Rejet qui ont transité par l'état statut Demandeur Vérifié auront différents types d'historiques d'exécutions qui dépendent du nombre d'exécution de la boucle CIC sur l'état Carrière Reconstituée, par exemples : $\delta 2 = ID.VI.CC.VE.NR$ pour les instances qui n'ont pas exécuté l'activité CIC, $\delta 3 = ID.VI.CC.CIC.VE.NR$, pour les instances ayant exécuté CIC une seule fois, ou encore $\delta 4 = ID.VI.CC.CIC.CIC.VE.NR$ pour les instances qui ont exécuté l'activité CIC deux fois, ...

Lors de l'application du pattern de substitution PM3, chaque instance aura un chemin d'exécution témoin spécifique, qui sera obtenu par une substitution adéquate. Par exemples :

- *l'historique d'exécution $\delta 2 = ID.VI.CC.VE.NR$ conduit au chemin d'exécution témoin $\beta 2 = ID.VI.VS.NR$, obtenu en remplaçant par VS le sous-chemin $CC.VE$ (conforme avec $\alpha_e = CC.v.VE$, quand v prend comme valeur le chemin vide ϵ).*
- *l'historique d'exécution $\delta 3 = ID.VI.CC.CIC.VE.NR$ conduit au chemin d'exécution témoin $\beta 3 = ID.VI.CIC.VS.NR$, obtenu en remplaçant le sous-chemin $CC.CIC.VE$ par $CIC.VS$, qui est une séquence conforme avec $\alpha'_e = v.VS$ (la variable v est égal à CIC). Le sous-chemin $CC.CIC.VE$ est une sequence qui satisfait $\alpha_e = CC.v.VE$ (la variable v prend la valeur CIC).*
- *l'historique d'exécution $\delta 4 = ID.VI.CC.CIC.CIC.VE.NR$ conduit au chemin d'exécution témoin $\beta 4 = ID.VI.CIC.CIC.VS.NR$, obtenu en remplaçant le sous-chemin $CC.CIC.CIC.VE$ (conforme avec $\alpha_e = CC.v.VE$ quand v prend la valeur $CIC.CIC$) par $CIC.CIC.VS$ (conforme avec $\alpha'_e = v.VS$ lorsque v est égal à $CIC.CIC$),*
- *...*

Il est observé que les chemins d'exécution témoins obtenus ($\beta 2$, $\beta 3$,$\beta 4$, ...), correspondent à des chemins d'exécution du nouveau protocole ***NRetraite*** *qui conduisent tous à l'état Rejet.*

6.5.2 Pattern imposant des obligations passées et futures

Après l'application des changements aux spécifications d'un protocole de service, les contraintes d'obligation peuvent être revues de manière à maintenir la cohérence globale des contrats exposés par le protocole. L'objectif et de faire migrer les instances

actives, tout en prenant en compte les exécutions passées, aussi bien que celles à venir. Cette action peut être réalisée en composant les patterns précédents d'obligations passées et futures.

A titre d'exemple, le pattern de substitution de sous-chemin **PM3**, imposant des obligations sur les exécutions passées, peut être combiné avec le pattern garantissant des obligations lors des interactions futures **PF2**. Cette composition conduit à un nouveau pattern composé ($PM3 \wedge PF2$), dont la sémantique intègre, simultanément, celle de **PM3** et celle de **PF2**. Ainsi, le nouveau pattern obtenu assure une stratégie de migration lors de la substitution d'un sous-chemin, en imposant en même temps l'exécution d'une séquence obligatoire d'activités dans le futur. Plus formellement :
Si $\alpha \in \alpha_e(\mathcal{P})$ est le sous-chemin à substituer par le sous-chemin $\alpha' \in \alpha'_e(\mathcal{P}')$ et si $\lambda \in \lambda_e(\mathcal{P}')$ est la séquence obligatoire d'activités à exécuter dans le futur, alors le pattern composé est exprimé par :

$$(\delta_e, q_s) \xrightarrow{\text{Subst } ((\delta,q_s),(\beta,q'_t),\alpha_e:\alpha'_e) \wedge \text{Obligatoire } ((\delta,q_s),(\beta,q'_t),\lambda_e)} (\beta_e, q'_t). \tag{12}$$

La sémantique véhiculée par le nouveau pattern est interprétée comme suit :
Le prédicat : $Subst\ ((\delta, q_s), (\beta, q'_t), \alpha_e : \alpha'_e) \wedge Obligatoire\ ((\delta, q_s), (\beta, q'_t), \lambda_e)$ est évalué à Vrai si : $\exists\ \delta^0, \delta^1, \delta'^0, \delta'^1 \mid (\delta = \delta^0\alpha\delta^1, \beta = \delta^0\alpha'\delta^1) \wedge (\forall w \in L(\mathcal{C}_{\mathcal{SP}'}(\mathcal{P}', q'_t)), w = \delta'^0\lambda\delta'^1)$.

L'application de la stratégie véhiculée par le pattern composé de l'équation (12) permet de gérer la migration d'une instance active i, ayant commencé son interaction dan l'ancien protocole et qui a exécuté un historique $\delta \in \delta_e(\mathcal{P})$. Cette stratégie stipule qu'un sous-chemin ($\delta = a.b.c$, par exemple) est remplacé par un autre sous-chemin $\delta' = d.e.f$, à condition qu'une séquence particulière d'activités *(par exemple : γ=Valider.Payer)* soit, obligatoirement, invoquée lors des interactions futures des instances transférées vers le nouveau protocole.

6.5.3 Exemple d'une stratégie combinant plusieurs patterns

La stratégie S2, spécifiée ci-après en utilisant les opérateurs logiques de conjonction et disjonction, illustre la composition des trois patterns de base PI1, PM1 et PM3. Elle permet d'appliquer une stratégie de migration plus élaborée qui intègre plusieurs types de contraintes à la fois.

$$(\delta_e, q_s) \xrightarrow{Condition1 \wedge [Condition2 \vee Condition3]} (\beta_e, q'_t) \tag{13}$$

où :
$Condition1$ = Interdit-P $((\delta, q_s), (\beta, q'_t), \gamma_e)$
$Condition2$ = Strict $((\delta, q_s), (\beta, q'_t), -)$,
$Condition3$ = Subst $((\delta, q_s), (\beta, q'_t), \alpha_e : \alpha'_e)$,
Les paramètres de la stratégie sont : $\delta_e = v$, $\beta = v'$, $\gamma_e = NR$, $\alpha_e = CC.v.VE^?$ et $\alpha'_e = v.VS$.

La stratégie S2 exploite le prédicat Interdit-P du pattern **PI1** *(Condition1)*, avec le paramètre $\gamma_e = NR$, pour sélectionner uniquement les instances qui n'ont pas encore exécuté l'activité NR. Ainsi, cette stratégie exclue du processus de migration les instances qui sont à l'état Rejet de l'ancien protocole Retraite. En plus, la stratégie S2 exige qu'une instance ayant un historique d'exécution δ, doit être transférée vers un

état q'_t du nouveau protocole NRetraite qui soit atteignable par un chemin d'exécution témoin β obtenu, soit par l'application du prédicat Strict du pattern **PM1** *(Condition2)*, exigeant que $\beta = \alpha$, ou par l'application du prédicat Subst du pattern **PM3** *(Condition3)*, avec les paramètres $\alpha_e = CC.v.VE^?$ et $\alpha'_e = v.VS$.

Le tableau 6.1 montre des exemples d'instances du protocole Retraite, avec leurs états et leurs historiques, respectives. L'application de la stratégie S2 aux instances permettra de les convertir en des instances du protocole évolué NRetraite. Les informations associées à l'application de cette stratégie sont données en spécifiant pour chaque instance, le chemin d'exécution témoin et l'état de migration cible.

Les deux premières lignes du tableau 6.1 correspondent à des instances qui sont migrées en utilisant le prédicat Strict, alors que les lignes restantes correspondent à des instances qui sont migrées par le biais du prédicat Subst.

Ancien Protocole (Retraite)		**Nouveau Protocol (NRetraite)**	
Etat Source	**Chemin d'exécution historique**	**Chemin d'exécution témoin**	**Etat Cible**
Demandeur Identifié	ID	ID	Demandeur Identifié
Vérification Terminée	$ID.VI$	$ID.VI$	Carrière Reconstituée
Carrière Reconstituée	$ID.VI.CC$ $ID.VI.CC.CIC$ $ID.VI.CC.CIC.CIC$ ...	$ID.VI.VS$ $ID.VI.CIC.VS$ $ID.VI.CIC.CIC.VS$ ...	Statut Demandeur Vérifié
Statut Demandeur Vérifié	$ID.VI.CC.VE$ $ID.VI.CC.CIC.VE$ $ID.VI.CC.CIC.CIC.VE$ ...	$ID.VI.VS$ $ID.VI.CIC.VS$ $ID.VI.CIC.CIC.VS$ ...	Statut Demandeur Vérifié
Demande Complète	$ID.VI.CC.VE.EDR$ $ID.VI.CC.CIC.VE.EDR$ ...	$ID.VI.VS.EDR$ $ID.VI.CIC.VS.EDR$ ...	Demande Complète
...	...	...	...

TABLE 6.1 – Exemples de migration d'instances avec la stratégie S2

A noter que d'autres types de composition de patterns peuvent être examinés. Par exemple, la combinaison du pattern d'obligations futures *(activités obligatoires : **PF2**)* avec le pattern d'interdictions passées *(activités interdites **PI1**)* conduira à un nouveau pattern qui exprime l'exécution conditionnelle d'une séquence d'activités (*si une séquence d'activités n'a pas été exécutée dans le passée, alors une autre séquence d'activités est obligatoire dans le futur*).

Avant de clôturer cette section, il est important de signaler que les stratégies de migration peuvent être appliquées d'une manière **séquentielle**. Par exemple, une nouvelle stratégie de migration S peut être définie en tant que séquence de stratégies S1 ;S2 ; où S1 est la stratégie décrite précédemment (*cf. sous section 6.5.1*) et S2 est la stratégie décrite ci-dessus. Dans ce contexte, les instances du protocole Retraite qui sont à l'état Rejet seront migrées en premier lieu conformément à la stratégie S1, après les instances restantes seront migrées conformément à la stratégie S2.

6.6 Etude de cas

Dans cette section, nous exposons une illustration complète de notre approche à travers une étude de cas, qui montre l'utilisation concrète des patterns de migration pour l'élaboration de différentes stratégies de migration. Ces stratégies sont déployées suite à l'évolution du protocole Retraite de la figure 5.1 vers une nouvelle version issue de l'application de plusieurs changements.

Nous commençons par présenter la nouvelle version du protocole Retraite obtenue suite à l'application de nouvelles lois et réglementations relative à la gestion des retraites. Après, nous spécifions des stratégies de migration simples et complexes, qui peuvent être déployées pour assurer la continuité d'exécution des instances actives dans le nouveau protocole. En fin, nous analyserons l'impact de l'évolution du protocole conformément aux stratégies de migration que nous avons spécifié.

6.6.1 Présentation du scénario d'évolution

Considérons le protocole de service donné dans la figure 6.4 qui illustre un nouveau protocole issu de plusieurs modifications. Ces modifications ont été appliquées au protocole Retraite initial de la figure 5.1 afin de l'adapter aux changements de la règlementation de la retraite.

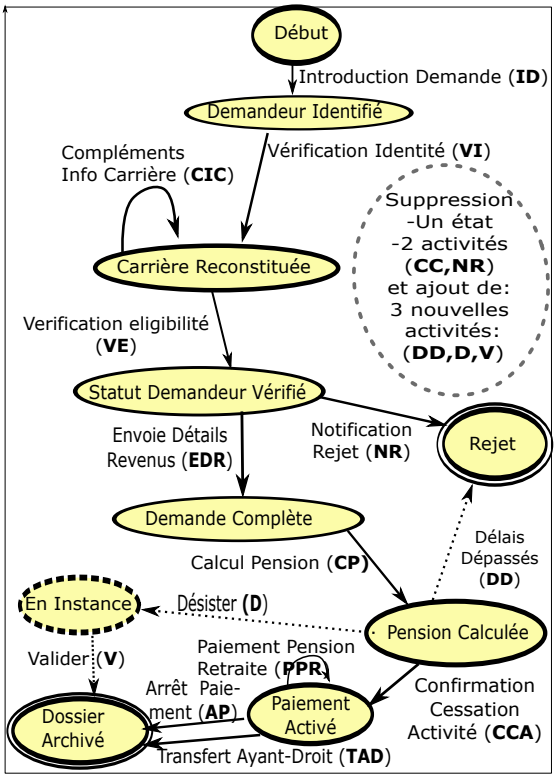

FIGURE 6.4 – Application des changements au protocole **Retraite** (**CRetraite**)

Description du nouveau protocole de service Dans le nouveau protocole de la figure 6.4, les modifications opérées apparaissent en pointillées et sont les suivants :

- Ajout d'un état : le nouvel état En Instance est ajouté au protocole initial.
- Ajout de trois transitions : une première transition, notée Désister (**D**), a été ajoutée pour permettre de transiter de l'état Pension Calculée vers le nouvel état En Instance. La deuxième transition (Valider (**V**)) est ajoutée pour passer du nouvel état En Instance à l'état final Dossier Archivé. Enfin, une troisième transition, notée Délais Dépassés (**DD**) est également ajoutée au protocole pour permettre aux instances de transiter de l'état Pension Calculée à l'état final Rejet du nouveau protocole.
- Suppression d'un état : l'état Vérification Terminée est supprimé du protocole initial.
- Suppression de deux transitions : les deux transitions Notification Rejet (NR) et Contrôle Carrière (CC), issues toutes les deux de l'état Vérification Terminée ont été supprimées. Ainsi, dans le nouveau protocole, l'exécution de l'activité Vérification Identité (VE) permet aux instances du service de transiter directement de l'état Demandeur Identifié vers l'état Carrière Reconstituée.

Les changements effectués sur la description du nouveau protocole prennent en charge de nouvelles règles métiers. En fait, les nouvelles lois régissant la retraite stipulent qu'un demandeur qui désire annuler ou reporter son départ à la retraite doit se désister. Ceci est possible tant que sa demande initiale est toujours en cours de traitement, et n'a pas encore atteint l'état Paiement activé. Cette action est réalisée par la nouvelle activité Désister qui permet de passer de l'état Pension Calculée à l'état En Instance. Néanmoins, le désistement ne sera effectif qu'après sa validation par le demandeur (*activité : Valider (V)*). Suite à la validation du désistement, le dossier sera archivé (*état Dossier Archivé*).

Une deuxième modification de la lois consiste à fixer un délais pour la transmission de la cessation d'activité vers l'organisme de gestion des retraites. En effet, une fois la pension calculée (*à partir de l'état Pension Calculée*), un délais légal (*6 mois, par exemple*) est accordé aux demandeurs afin d'envoyer leurs cessations d'activités (*activité Confirmation Cessation Activité (CCA)*) et permettre, ainsi, de fixer la date de jouissance de la pension. Passé ce délai, l'activité Délais Dépassés (DD) est déclenchée et le dossier sera rejeté, en faisant passer la demande à l'état **Rejet**.

Une dernière modification opérée sur le protocole Retraite est la suppression de l'état Vérification Terminée avec ses deux transitions sortantes. La première activité supprimée est Notification Rejet (NR) reliant les deux états Vérification Terminée et Rejet, et la deuxième est la transition Contrôle Carrière (CC) reliant l'état Vérification Terminée à l'état Carrière Reconstituée. Cette dernière actualisation du protocole est justifiée par des raisons de performance. En fait, l'organisme de retraite décide de reconstituer, au préalable, la carrière de tous les demandeurs identifiés, et la Notification Rejet (NR) ne sera opérée qu'une fois l'éligibilité contrôlée (*à partir de l'état : Statut Demandeur Vérifié*).

6.6.2 Spécification des stratégies de migration à appliquer

Au moment du changement du protocole de la figure 5.1, plusieurs instances sont actives et chacune a atteint un état particulier. Néanmoins, ces instances ont démarré leurs exécutions sur la base de l'ancienne spécification du protocole et, par conséquent, leur continuité dans le nouveau protocole de la figure 6.4 est compromise. Dans ce contexte, le problème consiste à *identifier les instances qui peuvent migrer vers le nouveau protocole et à assurer leur progression dans le cadre du protocole évolué de la figure 6.4*.

Pour répondre à cette préoccupation, le gestionnaire de protocole pourra exploiter les patterns de migration que nous avons formalisé précédemment, afin de mettre en œuvre différentes stratégies de migration qui prennent en charge cette préoccupation.

Le **tableau 6.2**, illustre des exemples de stratégies de migration permettant de gérer la migration des instances actives, suite à cette évolution (*chaque stratégie est identifiée par un numéro* **Num Stratégie**). Les deux premières stratégies exploitent des patterns de base, alors que les deux dernières utilisent la technique de composition.

Num Stratégie	Paramètres	Patterns
1	δ=ID.VI.CC α= CC β=ID.VI	PM4
2	δ=ID.VI.CC.(CIC)* α= CC β=ID.VI.(CIC)*	PM4
3	δ=ID.*.NR α= CC (pour PM4) α'= VE (pour PM5) β=ID.VI.VE.NR (pour PM4 et PM5)	PM4 ou PM5
4	δ=* ; α= CC ; $\mathsf{P}_1 = \mathcal{C}_{\mathcal{SP}}(Retraite, q_s)$; $\mathsf{P}_2 = \mathcal{C}_{\mathcal{SP}}(CRetraite, q'_t)$; β	(PM1 ou PM4) et PF3

TABLE 6.2 – Exemples de stratégies de migration à appliquer

6.6.3 Analyse de la migration conformément aux patterns

Nous analysons ci-après l'impact de l'application des différentes stratégies de migration du **tableau 6.2** sur les instances actives du protocole.

• Nous illustrons tout d'abord une utilisation simple du pattern **PM4** dont la spécification générique est rappelée ci-dessous :

$$(\delta_e, q_s) \xrightarrow{\text{Reduc } ((\delta,q_s),(\beta,q'_t),\alpha)} (\beta_e, q'_t)$$

La stratégie 1 (*i.e., ligne Num Stratégie=1 du tableau 6.2*) permet d'instancier ce pattern avec les valeurs des paramètres données dans la colonne 2 du même tableau. Cette stratégie a pour effet de transférer toutes les instances qui ont atteint l'état Carrière Reconstituée de l'ancien protocole, en suivant le chemin d'exécution δ=ID.VI.CC jusqu'à l'état Carrière Reconstituée. Le chemin d'exécution témoin dans le nouveau protocole est donné par : β=ID.VI (i.e., *suppression de l'activité initiale CC, activité qui n'existe plus dans le nouveau protocole*). Cependant, on peut observer que la stratégie 1 ne permet pas de transférer toutes les instances qui se trouvent à l'état Carrière Reconstituée de l'ancien protocole. En effet, les instances qui sont dans cet état et qui ont exécuté une ou plusieurs fois l'activité CIC ne seront pas pré-sélectionnées par la pré-condition de la stratégie 1.

• **La stratégie 2** (*ligne Num Stratégie=2 du tableau 6.2*) est plus générique que la stratégie 1, dans la mesure ou elle exploite l'opérateur de Kleene "*" pour déplacer toutes les instances qui se trouvent à l'état Carrière Reconstituée de l'ancien protocole vers l'état de même nom dans le nouveau protocole. Le transfert est opéré, tout en supprimant des anciens chemins d'exécution respectifs l'activité CC. Par exemple, un historique d'exécution *ID.VI.CC.CIC.CIC* de l'ancien protocole aura comme chemin d'exécution témoin ID.VI.CIC.CIC dans le nouveau protocole.

• **La stratégie 3** (*ligne Num Stratégie=3 du tableau 6.2*), dont la spécification est donnée ci-dessous, combine les patterns **PM4** et **PM5** à l'aide de l'opérateur de disjonction.

$$(\delta_e, q_s) \xrightarrow{\text{Reduc } ((\delta,q_s),(\beta,q'_t),\alpha) \vee \text{Extend } ((\delta,q_s),(\beta,q'_t),\alpha')} (\beta_e, q'_t)$$

L'instanciation des paramètres de la stratégie 3 est donnée dans le tableau 6.2. Cette stratégie permet de faire migrer toutes les instances qui se trouvent à l'état Rejet de l'ancien protocole vers l'état de même nom du nouveau protocole. Deux types de chemins d'exécutions témoins sont générés en fonction des chemins suivis par les instances pour atteindre l'état Rejet de l'ancien protocole. En effet, les instances ayant passé par l'état Carrière Reconstituée se verront appliqué le pattern **PM4** (i.e., suppression de l'activité CC de leurs historiques respectifs). Ainsi une instance qui a pour historique ID.VI.CC.VE.NR aura comme historique témoin dans le nouveau protocole ID.VI.VE.NR. Par contre, les instances ayant transité directement de l'état Vérification Terminée vers l'état Rejet dans l'ancien protocole, et qui ont donc comme historique d'exécution ID.VI.NR, se verront appliqué le pattern **PM5** (i.e., *extension de leur chemin d'exécution avec l'activité VE*) et auront comme historique d'exécution témoin ID.VI.VE.NR.

• **La stratégie 4** (*ligne Num Stratégie=4 du tableau 6.2*) illustre la combinaison des trois patterns **PM1**, **PM4** et **PF3**, à l'aide des connecteurs de conjonction et de disjonction. Elle permet de transférer une grande partie des instances actives en une seule fois, montrant ainsi la puissance d'expression des patterns proposés. Les valeurs des paramètres de cette stratégie sont données dans le tableau 6.2 et sa spécification est la suivante :

$$(\delta_e, q_s) \xrightarrow{[\text{Strict } ((\delta,q_s),(\beta,q'_t),-) \vee \text{Reduc } ((\delta,q_s),(\beta,q'_t),\alpha)] \wedge \text{Sous-protocol } ((\delta,q_s),(\beta,q'_t),(P_1:P_2))} (\beta_e, q'_t)$$

- Le prédicat Strict $((*, q_s), (\beta, q'_t), -)$ permet de transférer toute instance qui a atteint un état q_s de l'ancien protocole en suivant un chemin d'exécution quelconque δ, vers

l'état q'_t du nouveau protocole qui est atteignable par le chemin d'exécution témoin $\beta = \delta$.
- Le prédicat Reduc $((*, q_s), (\beta, q'_t), \text{Contrôle carrière})$ permet de transférer toute instance qui a atteint un état q_s de l'ancien protocole en suivant un chemin d'exécution quelconque δ vers l'état q'_t du nouveau protocole qui est atteignable par le chemin d'exécution témoin β, obtenue en supprimant l'activité Contrôle carrière du chemin δ.
- La combinaison de ces deux premiers prédicats avec une disjonction permet, donc, de transférer les instances de l'ancien protocole vers les états atteignables par le même chemin (*préservation stricte des exécutions*), ou bien par les chemins obtenus en supprimant l'activité Contrôle Carrière du chemin d'exécution considéré. Par exemple, les instances qui sont à l'état Vérification Terminée de l'ancien protocole, et qui ont un chemin d'exécution ID.VI seront transférées vers l'état Carrière Reconstituée du nouveau protocole, car cet état est atteignable par le même chemin d'exécution dans le nouveau protocole, suite à la suppression de l'état Vérification Terminée.
- Le prédicat Sous-protocol $((\delta, q_s), (\beta, q'_t), (P_1 : P_2))$, avec $P_1 = \mathcal{C}_{\mathcal{SP}}(Retraite, q_s)$ et $P_2 = \mathcal{C}_{\mathcal{SP}}(CRetraite, q'_t)$, rajoute une contrainte supplémentaire sur la migration en imposant que l'état q_s soit simulé par l'état cible q'_t. Cette nouvelle contrainte impose que le comportement possible à partir de n'importe quel état q_s peut être reproduit par l'état q'_t.
- La conjonction de ce dernier prédicat avec la première condition, permet à la stratégie 4 de faire migrer les instances de tous les états de l'ancien protocole sauf les trois états : Début, Demandeur Identifié et Vérification Terminée. Cette restriction vient du fait que l'état Vérification Terminée de l'ancien protocole, qui est atteignable par le chemin d'exécution ID.VI, n'est pas simulé par l'état Carrière Reconstituée qui est atteignable par le même chemin dans le nouveau protocole. En effet, l'activité NR est possible à partir de l'état Vérification Terminée de l'ancien protocole, mais elle n'est pas possible à partir de l'état Carrière Reconstituée du nouveau protocole.

A noter que les patterns d'interdiction d'activités passées (**PI1**) et futures (**PI2**) peuvent être utilisés à leur tour lors de la spécification des stratégies par combinaison de patterns. Dans cette perspective, la technique de composition de patterns permet de doter les gestionnaires de protocole d'un outil souple et efficace pour satisfaire leurs besoins spécifiques relatifs à la gestion de la migration d'instances.

6.7 Conclusion

Dans ce chapitre, nous avons présenté le deuxième aspect de notre approche déclarative de gestion de l'évolution des protocoles de services web. Il consiste en un enrichissement du langage de migration d'instances par l'identification et la formalisation d'un ensemble de patterns de migration.

Les patterns identifiés permettent la spécification de différentes stratégies de migration. Ils ont été exposés et illustrés par différents scénarios d'évolution. Néanmoins, les patterns identifiés ne sont pas exhaustifs. En effet, ils représentent ceux qui sont les plus **fréquemment** rencontrés lors des évolutions des protocoles de services. Pour dépasser cette restriction, nous avons proposé une technique de composition des patterns de base, afin de permettre aux gestionnaires de protocoles de spécifier des stratégies de migration plus élaborées.

Une étude de cas complète qui illustre l'application des différentes stratégies de migration a été présentée, et l'analyse de l'impact de l'évolution, suite à l'évolution du protocole Retraite, a été réalisée en exploitant les patterns proposés.

Il est à rappeler que les approches existantes pour la migration d'instances se focalisent, uniquement, sur la stricte mise en conformité des instances actives avec la nouvelle version du protocole (**pattern Strict (PM1)**). Dans notre approche, nous avons proposé une technique de migration plus flexible et qui est paramétrable.

Il convient de noter que, lors de la migration, le souci est de faire migrer le maximum d'instance actives. En ce sens, l'approche proposée apporte une plus value dans le domaine de la gestion des évolutions dynamiques des protocoles des services web. En effet, consolidée par un cadre formel solide, elle favorise la composition des patterns de base pour répondre aux besoins spécifiques lors de la gestion de la migration des instances actives.

Pour montrer la faisabilité de notre approche déclarative, nous abordons dans le prochain chapitre l'aspect implémentation et expérimentation.

Chapitre 7

Implémentation et expérimentation

7.1 Introduction

Afin de valider les contributions proposées tout au long de ce manuscrit, nous avons mis en œuvre un prototype logiciel qui intègre différents composants implémentant chacun des aspects évoqués dans notre approche, puis nous avons expérimenté les performances de l'outil développé.

Dans ce chapitre, nous présentons et nous décrivons de manière succincte le prototype expérimental, nommé **P**rotocol **C**hange **I**mpact **A**nalyser **(PCIA)** qui, implémente notre approche déclarative de migration d'instances. Différents expérimentations ont été réalisées sur des données synthétiques, pour évaluer le passage à l'échelle et les performances du prototype réalisé. En fin, nous avons évaluer la capacité des patterns de migration proposés à capturer les principales stratégies de migration existantes dans la littérature et nous avons examiné leur aptitude à prendre en charge les différents critères de conformité répandus dans le domaine de la gestion des évolutions.

Ce chapitre est organisé comme suit : la section 7.2 présente la conception, l'architecture générale et les fonctionnalités du prototype **PCIA** ainsi que la description de chacun des modules qui le compose. La section 7.3 décrit en détails différents scénarios d'utilisation du prototype réalisé pour gérer les tâches inhérentes à l'évolution. Quelques résultats expérimentaux sur des données synthétiques sont présentés et discutés en section 7.4. En fin, une évaluation de l'expressivité et de la complétude de l'approche proposée est discutée en section 7.5. La section 7.6 récapitule le chapitre et discute brièvement les futures perspectives de développement de notre prototype.

7.2 Architecture et fonctionnalités du prototype

Dans cette section, nous exposons la mise en œuvre de notre approche. On commence par exposer le modèle de données conçu et nous illustrons comment les informations manipulées sont stockées dans des structures de données adéquates. Après, nous présentons l'architecture du système développé et nous terminons la section par l'énumération des fonctionnalités offertes par le prototype **PCIA**.

7.2.1 Modélisation des structures de données

Toutes les données manipulées par le système lors de l'évolution des protocoles des services sont capturées et stockées dans des structures de données adéquates. Ainsi, les informations relatives à la définition, à la création, à l'invocation par les clients et à la gestion des changements des protocoles sont modélisées dans des structures de données intégrées. En effet, nous modélisons le domaine de l'évolution des protocoles

et de l'analyse de son impact par un diagramme de classe **UML** qui sera transcrit dans une base de données relationnelle correspondante. Ce modèle offre une représentation abstraite des objets traités par le système et qui interagissent pour réaliser différentes fonctions répondant aux cas d'utilisation.

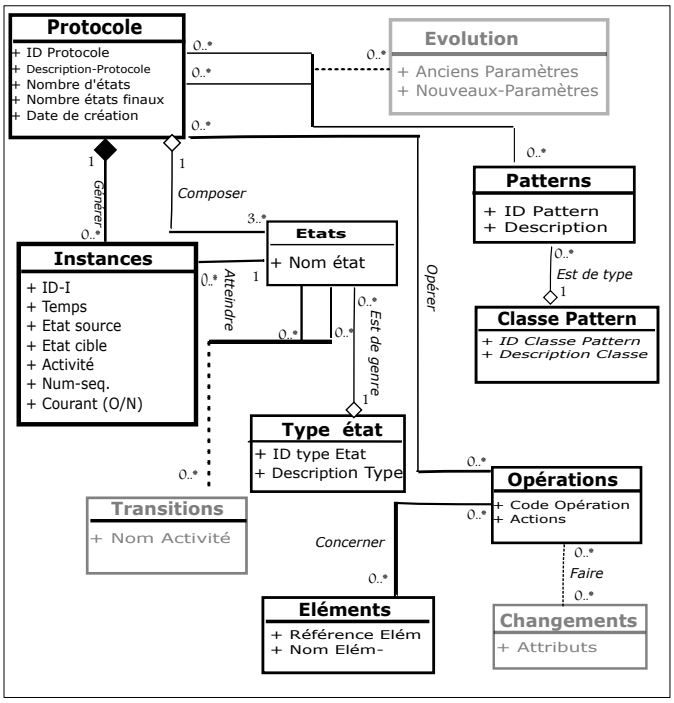

FIGURE 7.1 – Diagrame UML modélisant la gestion de l'évolution des protocoles

Le diagramme UML, illustré dans la figure 7.1, prend en charge toutes les données afférentes au cycle de vie des protocoles de service et à la gestion de la migration des instances, à savoir :

1. **La description des protocoles** : la classe Protocole est décrite avec différent attributs exprimant l'identification du protocole, sa description textuelle, le nombre d'états et d'états finaux et sa date de création. Cette classe est en association avec la classe Etats qui détaille chaque état du protocole. Pour chaque état, son type est décrit en associant la classe Etats avec la classe Type état qui énumère les différents types d'états possibles *(initial, intermédiaire, final)*. L'association Est de genre fournit cette classification. Les transitions entre les états du protocole sont exprimées par la classe composée Transitions qui spécifie le nom de l'activité associée à chaque transition entre deux états quelconques.
2. **Stockage des instances d'exécutions** : lors de l'invocation d'un service par les instances, toutes les activités exécutées sont enregistrées dans la classe Instances. Ainsi, cette classe fournit les séquences d'activités associées aux instances (*les traces*), depuis l'invocation d'un service par l'instance ; (association Générer) jusqu'à son état actuel. La classe Instances est reliée à la classe Etats (*association Atteindre*) pour capturer l'état courant atteint par chaque instance. L'attribut Courant (O/N) de cette classe renseigne sur le type de l'instance (*active ou terminée*). Si l'état courant de l'instance est un état final alors l'instance est terminée.

D'autres informations relatives au temps et aux activités exécutées sont stockées dans la même classe Instances (*temps début, état source, état cible, ...*).

3. **Gestion de l'évolution du protocole** : le modèle de données conçu gère les changements opérés sur les descriptions des protocoles, en stockant les opérations de modifications opérées (*classe : Changements*). La nature des changements opérés *(ajout, suppression, renommage, ...)* est décrite par l'attribut Actions de la classe Opérations. Par le biais de l'association Concerner, cette dernière classe prend en charge les changements appliqués aux éléments de la structure du protocole *(états, messages, chemins, sous-protocoles,...)* contenues dans la classe Eléments. La classe composée Evolution gère les paramètres de la migration (attributs :*anciens paramètres, nouveaux paramètres*) associés aux patterns de migration (classe Patterns). En fin, la classe Classe Pattern identifie la catégorie du pattern *(exécutions passées, exécutions futures, obligation, interdiction)*.

Une fois créée est alimentée, le scénario typique d'exploitation de la base de données passe par les deux étapes principales suivantes :

- Interroger la base de données pour sélectionner et récupérer les informations relatives aux instances (*chemins d'exécution, état atteint*), et choisir le pattern de migration à activer pour déployer la stratégie de migration. Ces informations sont utilisées comme des entrées du système.
- La base de données est explorée pour récupérer les attributs correspondants à la migration des instances actives sélectionnées, conformément au pattern de migration en question (*état de migration, trace témoin*).

7.2.2 Stockage et interrogation des données

A divers niveaux de notre application, les besoins de stockage, d'accès et de gestion des informations relatives aux instances et aux protocoles nécessitent un langage d'interrogation de bases de données. Comme expliqué précédemment, le prototype **PCIA** utilise un système de gestion de base de données relationnelle, et le langage déclaratif **SQL** est exploité comme outil de base pour l'implémentation des patterns de migration. Notre choix des bases de données relationnelles et du langage d'interrogation **SQL** sont motivés par les trois raisons principales suivantes :

1. **Facilité d'utilisation :** les bases de données relationnelles sont reconnues pour leur popularité et leur simplicité pour les utilisateurs finaux qui sont familiarisés avec les langages déclaratifs comme les requêtes SQL. Du point de vue pratique, la base de données est indexée sur des attributs pertinents pour faciliter son exploitation lors de la recherche des traces témoins et des états de migration. Ainsi, les développeurs et les gestionnaires de protocoles peuvent interroger cette base de données à la recherche d'informations (*états, traces, ...*) de façon déclarative ou via l'interface graphique et conviviale du système (**GUI**).
2. **Intégration facile avec les systèmes BPM existants :** La plupart des systèmes **BPM** fournissent des moyens pour la persistance des données d'exécution (*généralement en utilisant la spécification : Java Persistence API*). En plus, les systèmes **BPM** existants sont agnostiques aux **SGBD** et permettent aux utilisateurs de configurer n'importe quelle source de données ils vont utiliser pour

gérer les données d'exécution. Dans cette optique, il est observé que dans la plupart des systèmes existants, à la fois les informations sur les définitions des protocoles de services et processus métiers ainsi que celles relatives à leurs exécutions (*par exemple, les données des instances en cours*) peuvent être stockées dans une base de données relationnelle intégrée au système **BPM**.

3. **Besoins de performances** : Les systèmes natifs de gestion des bases de données orientées graphes (*par exemple, Neo4J [167] utilise le langage Cypher [168] et OpenLink Virtuoso*[1] *utilise le langage SPARQL [169]*) utilisent des langages déclaratifs de requêtes sur des graphes. Cependant, les travaux récents de la communauté des bases de données [170], montrent empiriquement que pour les tâches d'analyse et d'exploration de graphes, il est plus efficace de s'appuyer sur un moteur relationnel que sur un moteur spécialisé de graphes. Les mêmes résultats et observations empiriques ont été confirmés dans une étude très récente qui compare les performances des **SGBD** relationnels à usage général par rapport à celles des moteurs natifs de graphes, lors de l'évaluation des requêtes récursives et non récursives sur des graphes [171, 172].

En se basant sur les motivations précédentes, l'implémentation actuelle du prototype **PCIA** utilise le schéma de la base de données propriétaire résultant du diagramme de classe de la figure 7.1 pour stocker les informations sur les instances et les protocoles des processus métiers. Cependant, nous signalons que le prototype implémenté peut être facilement adapté à la plus part des systèmes de gestion des bases de données des environnements **BPM** existants.

Le tableau 7.1 montre un extrait de la table Instances de la base de données conçue. Elle stocke les différentes données associées aux instances des protocoles.

ID-I	Temps	Etat source	Etat cible	Activité	Num Séq.	Courant
i_1	t_0	Début	Demandeur Identifié	ID	0	Non
i_1	t_1	Demandeur Identifié	Vérification Terminée	VI	1	Non
i_2	t_2	Début	Demandeur Identifié	ID	0	Non
i_1	t_3	Vérification Terminée	Carrière Reconstituée	CC	2	Non
i_1	t_4	Carrière Reconstituée	Carrière Reconstituée	CIC	3	Non
i_2	t_5	Demandeur Identifié	Vérification Terminée	VI	1	Non
i_1	t_6	Carrière Reconstituée	Carrière Reconstituée	CIC	4	Oui
i_2	t_7	Vérification Terminée	Carrière Reconstituée	CC	2	Oui
i_3	t_8	Début	Demandeur Identifié	ID	0	Non
i_3	t_9	Demandeur Identifié	Vérification Terminée	VI	1	Non
i_3	t_{10}	Vérification Terminée	Rejet	NR	2	Oui
...	...	...	...	...	...	...

TABLE 7.1 – Un extrait de la table Instances de la base de données relationnelle

Le tableau contient des informations relatives à trois instances dont deux seulement sont en cours d'exécutions (*instances actives*). L'instance i_1, ayant la trace d'exécution ID.VI.CC.CIC.CIC (*instance active*), i_2 dont la trace d'exécution est : ID.VI.CC (*instance active*) et i_3 qui a pour trace d'exécution ID.VI.NR (*instance terminée ; i.e : elle a atteint l'état final Rejet*) .

1. https ://virtuoso.openlinksw.com/

Chaque tuple dans la relation stocke les informations sur les exécutions des activités du protocole par les instances, à savoir : l'identifiant de l'instance (ID-I), le temps d'exécution correspondant au temps auquel l'activité a été déclenchée (Temps), l'état source de la transition (Etat source), l'état cible après l'exécution de l'activité (Etat cible), le nom de l'activité invoquée (Activité), le numéro de séquence de l'activité (Num Séq.) correspondant à l'ordre de cette activité dans la trace d'exécution de l'instance considérée, et finalement si oui ou non il s'agit de la dernière activité exécutée par l'instance. Dans le cas positif, l'attribut Etat cible de l'instance n'est autre que son état courant. Alors, l'attribut (Courant) est positionné à la valeur Oui.

D'autres contraintes d'intégrité sont implémentées pour garantir que la cohérence de la structure du protocole (*i.e., la définition du protocole, états atteignables, existence d'états finaux...*) est préservée dans la base de données. De cette façon l'implémentation de certains patterns de migration est considérablement facilitée.

A titre d'illustration, considérons le pattern de migration Strict de l'exemple 6.2 :

$$(\delta_e = v, q_s) \xrightarrow{\text{Strict } ((\delta,q_s),(\beta,q'_t))} (\beta_e = v', q'_t).$$

Ce pattern permet de convertir toute instance i, dotée d'un chemin d'exécution δ dans l'ancien protocole, en une instance i' compatible dans le nouveau protocole seulement s'il existe un état q'_t dans ce dernier qui peut être atteint en utilisant le même chemin d'exécution δ.

Une mise en œuvre simple de ce pattern dans notre système consiste en une copie en bloc de la table Instances de l'ancien protocole dans la table Instances du nouveau protocole, tout en filtrant les seules instances actives lors du transfert. Ainsi, les instances de l'ancien protocole qui ont un chemin d'exécution δ qui n'est pas conforme avec les spécifications du nouveau protocole sont automatiquement écartées de la table Instances du nouveau protocole, à cause de la violation des contraintes d'intégrité qui assurent la consistance des instances par rapport à la définition du protocole.

Les autres patterns, particulièrement, ceux qui manipulent des variables de chemins, ne sont pas entièrement déclaratifs et nécessitent des implémentations plus complexes qui intègrent des requêtes **SQL** dans un langage de programmation complet (*dans notre cas, le langage Java*). D'une manière générale, l'implémentation des différents patterns de migration proposés profite de la disponibilité de compilateurs qui sont capables de convertir des requêtes sur des graphes en de requêtes **SQL** correspondantes [173, 174].

7.2.3 Architecture globale du prototype réalisé

La figure 7.2 montre l'architecture du prototype **PCIA** et les interactions entre ses éléments. Comme il est observé dans la figure, le système est organisé autour de quatre composants principaux qui interagissent avec la base de données. Il est implémenté en Java dans l'environnement de développement intégré (**IDE**) Eclipse. Le choix d'Eclipse a été dicté par le fait que c'est un **IDE** open source largement utilisée pour construire des plates-formes de développement ouvertes et extensibles constituées d'outils et d'**API** destinés à la construction, au déploiement et à la gestion des logiciels.

Les composants développés interagissent avec la base de données relationnelle conçue précédemment et deux API ont été utilisées pour la manipulation des données par le prototype **PCIA**. La première est l'API **JDOM** [175, 176] permettant la manipulation (*création, modification, suppression, ...*) des documents XML via une structure arborescente. Et la deuxième est l'API **mysql-connector** [177] qui offre un accès aux bases de données sous java.

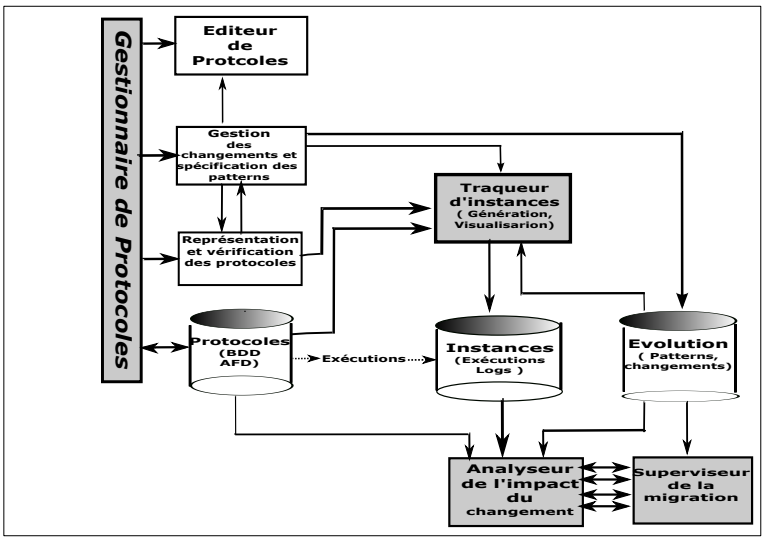

FIGURE 7.2 – Interactions entre les composants du système.

Les composants du système sont brièvement décrits ci-après.

- **Un gestionnaire de protocoles** constitué de trois modules. Le premier module est un éditeur graphique de protocoles (Éditeur de protocoles) qui permet aux utilisateurs de créer et de modifier la description des protocoles via une interface graphique dotée d'une boite à outils conviviale. Le protocole construit est transcrit en un fichier XML et stocké dans la base de donnée Protocoles. Le deuxième module est dédié à la gestion des changements et à la spécification des patterns. Il permet de créer les patterns de migration et leurs éléments descriptifs, suite à une évolution d'un protocole. Les données générées sont enregistrées dans la base de données Evolution. Le troisième module (Représentation et vérification des protocoles) fournit différents modèles de représentation des protocoles (*formel, graphique, XML*) qui sont très utiles pour la vérification des anomalies de spécification (*états non atteignables, états finaux, ...*).
- **Un traqueur d'instances** qui assure la génération et le stockage des traces d'exécution des instances dans la base de données Instances, telle que décrite précédemment (cf., 7.2.2, table 7.1). En plus, il offre la possibilité aux gestionnaires de protocoles de simuler des exécutions des protocoles, en générant des instances d'une manière manuelle ou aléatoires. Par la suite, plusieurs possibilités de visualisation des instances sont offertes aux utilisateurs finaux (*par protocole, par état atteint, instances actives durant une période donnée, ...*).

- **Un analyseur de l'impact du changement** qui constitue le noyau du système. Il assure l'analyse de l'impact des changements sur les instances actives. Concrètement, pour une ou plusieurs instances actives en entrée, il calcul les traces témoins correspondantes et les états de migration associés, conformément à la stratégie de migration adoptée par l'utilisateur. Cette analyse peut être opérée individuellement ou sur un ensemble de traces sélectionnées.
- **Un superviseur de la migration** c'est un outil d'aide à la prise de décision qui assiste le gestionnaire de protocoles dans les tâches de maintenance et d'évolution. Ce module produit différents indicateurs relatifs à la gestion de la migration, tels que : *les taux de migration, le temps écoulé pour réaliser la migration, nombre d'instances non migrables, ...*

7.2.4 Fonctionnalités du prototype réalisé

L'exploitation du prototype **PCIA** permet aux gestionnaires de protocoles de réaliser les tâches suivantes de gestion et d'analyse de l'impact de l'évolution des protocoles.

- Décrire graphiquement la spécification du protocole .
- Mettre à jour et visualiser les protocoles dans différents formats (*XML, graphique, formel*).
- Opérer les changements des descriptions des protocoles et spécifier les patterns de migration lors des évolutions.
- Générer manuellement ou aléatoirement des traces d'exécutions et les stocker dans la base de données.
- Visualiser les instances actives (*ou terminées*) et leurs états courants dans un protocole donné.
- Réaliser des analyses de l'impact du changement sur les instances actives, conformément aux patterns appliqués. Les données de migration *(les traces d'exécution témoins, les états correspondant à la migration)* seront fournis par le système.
- Superviser les évolutions des protocoles et exploiter les données statistiques (*taux et temps de migration, ...*)

7.3 Scénario d'exploitation du système

Dans cette section, nous présentons l'enchainement général des fonctions offertes par le prototype réalisé. Les scénarios d'utilisation sont illustrés par différentes captures d'écrans récupérées à partir du système **PCIA**. Ils couvrent les différentes fonctions depuis la création initiale des protocoles jusqu'à l'analyse de l'impact et la gestion de la migration des instances actives.

7.3.1 Édition d'un protocole

Le système est doté d'un éditeur graphique destiné à créer, modifier et visualiser les descriptions des protocoles. La figure 7.3, représente l'interface d'édition d'un automate décrivant un protocole de service. A l'ouverture de l'application, l'interface est dépourvue de contenue à part la barre de menu. Pendant les opérations de mise à jour

(*items du sous-menu* Protocols Editor), une boîte à outils contenant des éléments graphiques (*états, messages, ...*) est griffée à l'interface. Cette boite à outils sert à éditer les protocoles (*création ou modification*). Les actions permises consistent à choisir le type d'états (*initial, intermédiaire, final*) ou à choisir les transitions (*activités*), et de procéder à leur description (*éventuellement leur suppression*). Dans la barre d'outils, en dessous de la barre de menu, figurent des icônes pour l'accès rapide aux opérations les plus fréquentes : création de protocole, ouverture d'un protocole en modification ou enregistrement d'un protocole,...

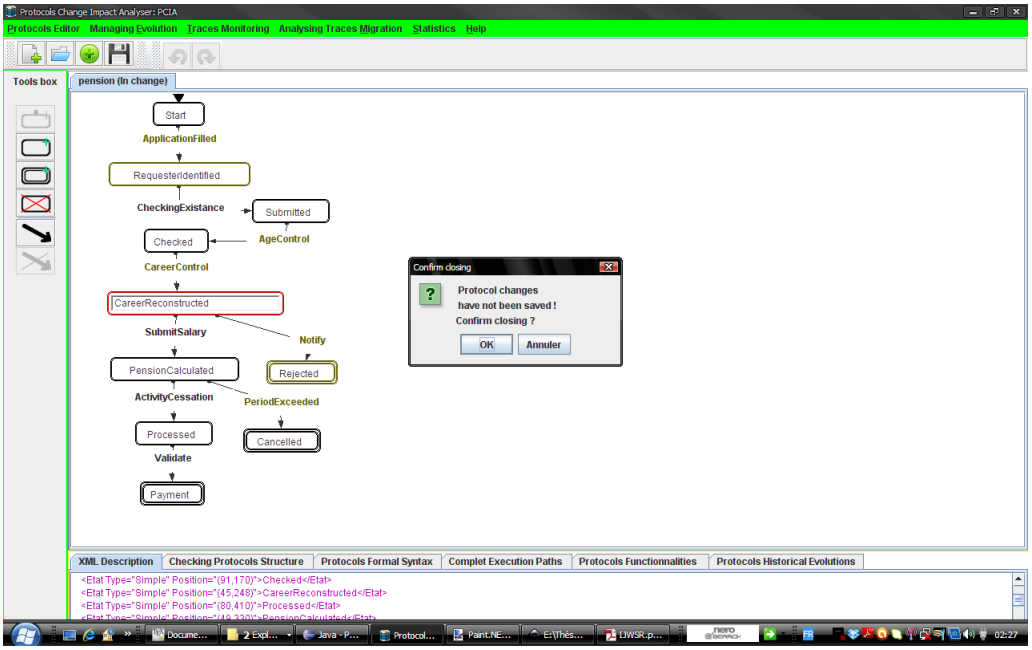

FIGURE 7.3 – Création, modification et visualisation des protocoles

Durant les opérations de création ou de modification des protocoles, un analyseur syntaxique vérifie la conformité du modèle par rapport aux critères formels. En effet, toutes les erreurs de description d'un automate, telles que l'absence d'états finaux, états non atteignables ..., sont signalées aux concepteurs du protocole. D'autre part, le système offre aux utilisateurs différentes représentations des protocoles, à savoir : la représentation en syntaxe formelle (*conformément à la définition 5.1*), la représentation en XML, les différents chemins d'exécution existants, la description textuelle du protocole ainsi que la liste de ses versions antérieurs générées lors des évolutions. Ces informations, exposées dans le panneau de la zone en bas de l'interface, sont très utiles aux développeurs de protocoles.

7.3.2 Gestion des changements et spécification des patterns

Lors du chargement d'un protocole existant pour opérer des changements, (*menu* Managing evolution → protocol evolution), l'automate correspondant à la dernière version est directement visualisé en plein écran sous forme graphique. Par ailleurs, si ce protocole a subi des évolutions antérieures de sa description alors la liste des versions

précédentes est affichée dans le dernier onglet du panneau de bas, tel que illustré dans la figure 7.4.

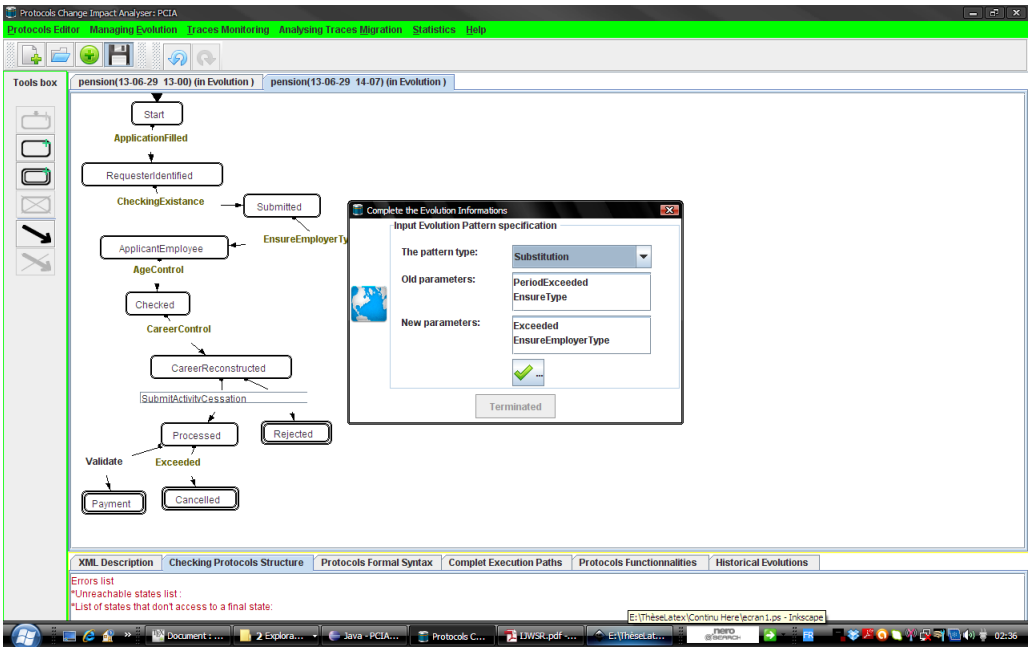

FIGURE 7.4 – Spécification des patterns de migration lors de l'évolution d'un protocole

Une fois les changements de la description achevés, une boite de dialogue permet d'introduire les types de patterns de migration associés à cette évolution (PM1,. . . ,PM5, PF1,. . . ,PF3, PI1,. . . ,PI2). Pour chaque pattern ses paramètres descriptifs sont requis (*voir figure 7.4*). Les informations relatives à l'évolution et aux patterns sont stockées dans les bases de données respectives (Protocole, Evolution et Patterns) et serviront de paramètres lors de l'élaboration des stratégies de migration.

7.3.3 Exécution des protocoles et génération des traces

La capture d'écran de la figure 7.5, représente l'interface de génération des traces d'exécution des protocoles.

Après le chargement du protocole dont on désire simuler l'exécution, l'automate associé est visualisé à gauche de l'écran et sa dernière version est affichée au milieu de l'écran. Deux modes de génération de traces sont proposés aux utilisateurs : **manuelle** et **aléatoire**. La génération manuelle, consiste à sélectionner les activités à exécuter, une par une, à partir de la liste des opérations permises par le protocole chargé. Quant à la génération aléatoire, elle exige uniquement l'introduction du nombre total de traces à générer. Puis en utilisant ce paramètre, le système applique une fonction aléatoire de parcours des chemins possibles du protocole en question, tout en invoquant les opérations contenues dans les différents chemins. Les traces générées peuvent être visualisées selon différents critères d'affichage (*par état atteint, par chemins parcourus, par période de création, . . .*). En plus de la génération des traces, la possibilité est donnée à l'utilisateur pour supprimer celles qui sont jugées incohérentes ou inutiles.

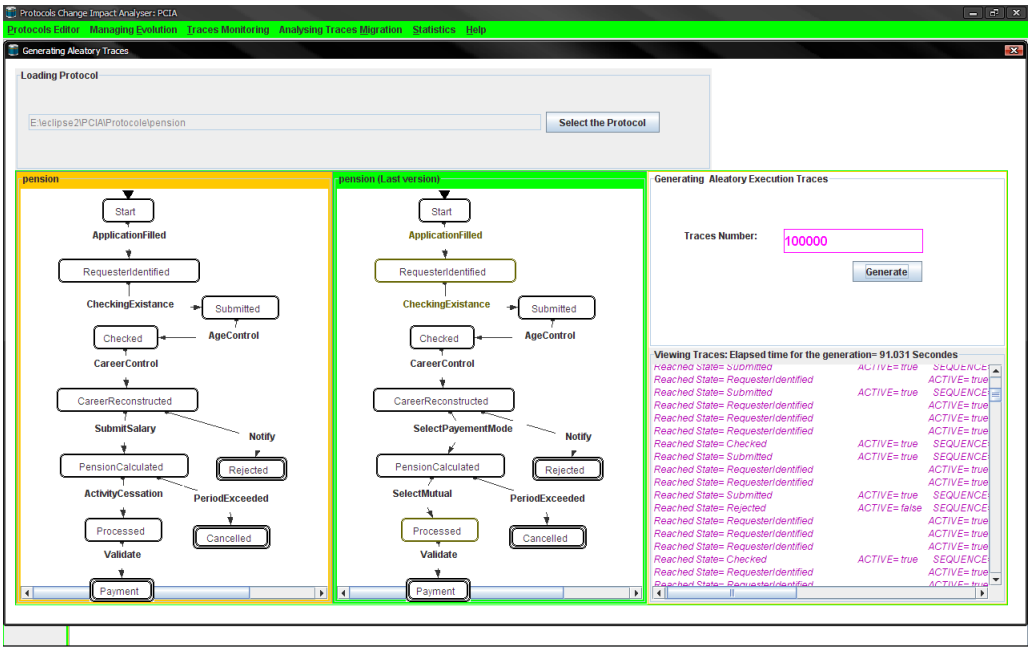

FIGURE 7.5 – Génération des traces d'exécution

7.3.4 Analyse de l'impact des changements

A présent, tous les ingrédients sont disponibles pour pouvoir faire l'analyse de l'impact de l'évolution et assurer la migration des instances actives d'un protocole qui a subi des évolutions dans sa description.

Le gestionnaire de protocoles commence par filtrer les instances actives du protocole concerné, puis il déclenche le processus de leur migration, conformément à la nouvelle version du protocole et sur la base de(s) pattern(s) de migration spécifique(s) ou leur composition. L'interface de la figure 7.6, illustre un tel scénario.

Tout d'abord, le gestionnaire de protocoles charge la version initiale du protocole, après il sélectionne la version du protocole cible de la migration. Une fois les instances actives affichées, il filtre celles qui sont concernées par la migration, puis il spécifie sa stratégie de migration (*basée sur un simple pattern ou guidée par une stratégie composée de patterns*). A cette étape, le processus de migration peut être actionné par un clic sur le bouton Migrating (*en bas à gauche de la figure 7.6*). Pour chaque instance sélectionnée, l'identifiant de l'instance, la séquence d'opérations (*trace*), l'état atteint et la décision sur la migration (**oui/non**) sont affichés à gauche de l'écran. Après achèvement du processus de migration, les informations associées à la migration, à savoir : les traces témoins correspondantes et les états cibles dans le nouveau protocole sont affichées à droite de l'écran. Par ailleurs, différentes données statistiques sont fournies aux utilisateurs du système (*taux de migration, temps écoulé pour la migration, ...*).

Une fois la migration terminée, les instances qui ont été transférées vers le nouveau protocole sont dotées de nouvelles traces témoins et pourront continuer leur progression dans le contexte de ce dernier.

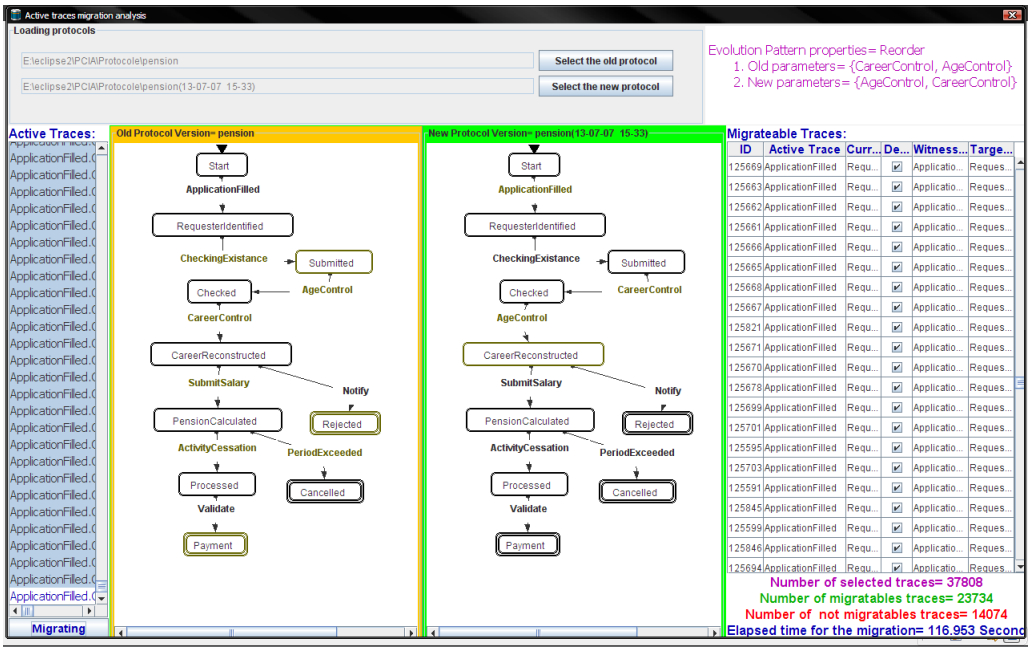

FIGURE 7.6 – Analyse de la migration des instances actives vers le nouveau protocole

7.4 Expérimentation

Pour évaluer le passage à l'échelle et les performances du prototype implémenté, différents ensembles de données synthétiques ont été expérimentés en utilisant le le système **PCIA**. Les expérimentations menées visent à montrer l'efficacité et l'importance des patterns de migration identifiés. Les étapes du protocole expérimental sont les suivantes : (i) dans un premier temps, nous avons généré des ensembles de données de différentes tailles pour servir de jeu de test de l'expérimentation. (ii) ensuite, nous avons calculé et comparé le temps de migration pour chaque pattern et chaque ensemble de données. (iii) et en fin, nous avons évalué le temps et les taux de migration des instances et nous avons analysé les corrélations entre les éléments du système.

7.4.1 Préparation des données d'expérimentation

Pour la génération des données d'expérimentation, nous avons procédé à la simulation des exécutions du protocole Retraite de la figure 5.1. Quatre ensembles de données (*data-sets* : **DS**) synthétiques contenant, respectivement, $DS1 = 1.000$, $DS2 = 10.000$, $DS3 = 100.000$ et $DS4 = 1.000.000$ instances ont été générées de manière aléatoire et leurs traces d'exécution sont sauvegardées dans la base de données des Instances , décrite dans la section 7.2.2 (*table. 7.1*).

Comme illustré dans la figure 7.7, il est observé que le temps de génération suit une évolution croissante en fonction du nombre d'instances à générer. Ce temps atteint 1060 secondes pour l'ensemble de données le plus large *DS4* ($\simeq$ *17 minutes*).

Une fois les instances sont générées et sauvegardées, uniquement celles qui sont à l'état actives seront filtrées pour le processus d'expérimentation, car les instances terminées sont sans aucun intérêt dans notre contexte de migration d'instances actives.

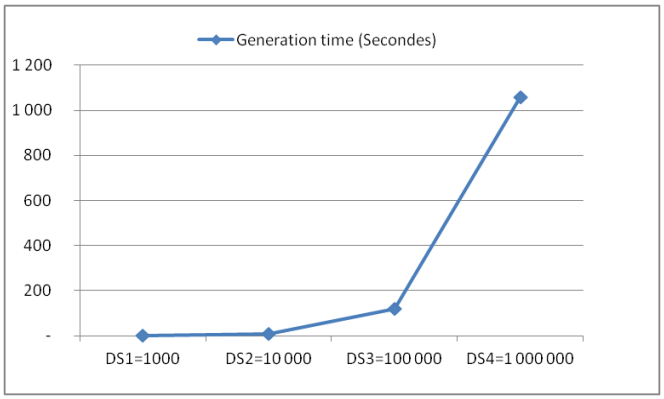

FIGURE 7.7 – Evolution du temps de génération des données d'expérimentation

Afin de s'assurer de la consistance du nombre d'instances actives et de la représentativité des échantillons de test, nous avons calculer les taux des instances actives par rapport au nombre total des instances générées.

La figure 7.8 montre que ce taux est légèrement supérieur à 88 % pour l'ensemble des échantillons manipulés et qu'il suit une croissance non linéaire, à cause de la fonction aléatoire de génération des instances. Néanmoins, ce taux est largement satisfaisant pour mener les analyses de l'impact lors de la migration des instances actives.

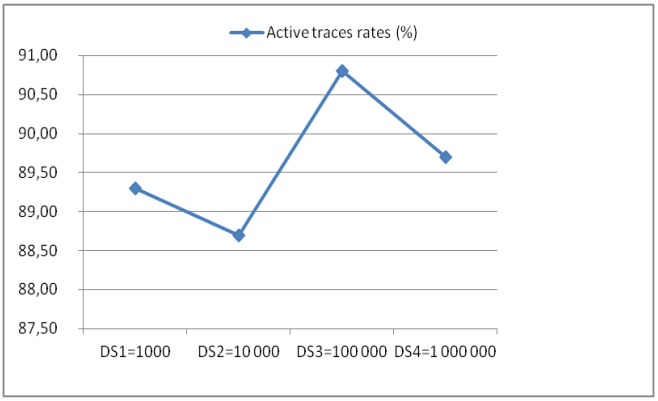

FIGURE 7.8 – Variation des taux des instances actives

7.4.2 Tests de passage à l'échelle

Pour examiner le passage à l'échelle du prototype réalisé, une première expérimentation a été conduite en exploitant le pattern de migration Strict (**PM1** ; voir section 6.3.1). Dans cette perspective, le temps nécessaire, pour accomplir la migration des différents lots d'instances conformément à ce pattern, a été calculé et analysé.

La figure 7.9 illustre la variation du temps de migration pour le pattern Strict. Il est observé que ce temps croit exponentiellement en fonction du volume d'instances à

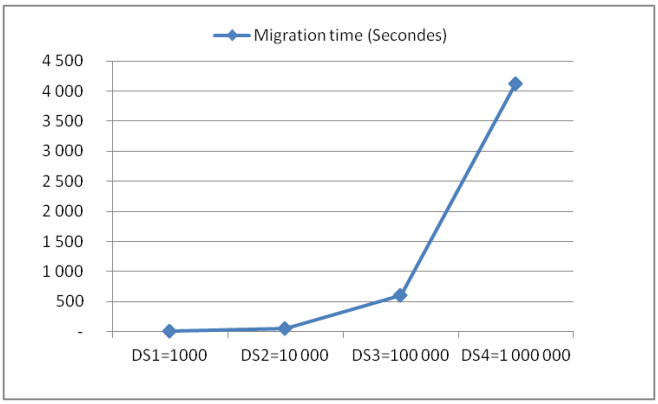

FIGURE 7.9 – Variation du temps de migration pour le pattern Strict **PM1**

faire migrer de l'ancienne version du protocole Retraite vers sa nouvelle version.

Comme le processus de migration doit être opéré *(off-line)*, les résultats obtenus sont encourageants *(4036 secondes ≃ 1 heure pour faire migrer un million d'instances du lot **DS4**)*. Cette observation motive l'exploitation du système pour la prise en charge des applications grand-public.

Dans une deuxième expérience, le temps nécessaires pour faire migrer les instances actives, conformément aux différents patterns de migration, a été étudié. Cette expérience vise à comparer l'ordre de grandeur du temps de migration pour les différents patterns. Pour cela, un ensemble de données de grande taille a été choisi ($DS3 = 100.000$ instances), puis nous avons fait varier les patterns de migration d'obligations historiques (**PM1,. . . , PM5**), tout en calculant le temps de migration pour chaque lot et chaque pattern.

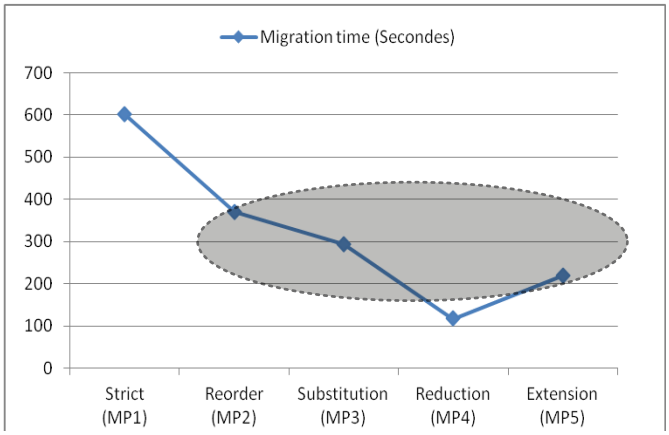

FIGURE 7.10 – Temps de migration du lot **DS3 (100.000)** pour les différents patterns

Le graphe de la figure 7.10 montre que ce temps varie de 603 secondes pour le pattern (Strict) et descend jusqu'à 112 secondes pour le pattern (Réduction **PM4**). Pour les autres patterns, le temps de migration est du même ordre de grandeur (*entre 200 et 400 secondes, voir zone ombrée)*. Ces résultats sont justifiés par le fait que pour

le pattern (Strict), une comparaison exacte est faite pour chercher la trace témoin d'une instance active. La mise en œuvre de ce pattern exige une copie en bloc de la table *Instances* de l'ancien protocole vers la table *Instances* du nouveau protocole, telle que expliqué dans la section 7.2.2. Cette opération engendre un temps de traitement considérable. Cependant, pour le dernier pattern (Réduction **PM4**), une sous-trace est, tout simplement, supprimée de la trace de l'instance à faire migrer.

7.4.3 Évaluation des performances du système

Pour évaluer les performances du prototype PCIA, nous avons calculer le temps total de migration des quatre ensembles de données de test, et ce conformément aux cinq patterns d'obligations historiques (**PM1,. . . , PM5**).

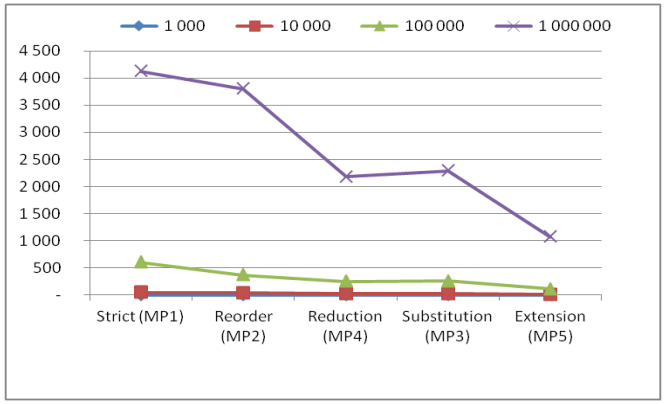

FIGURE 7.11 – Analyse du temps de migration pour les divers patterns

Les résultats exposés dans la figure 7.11, montrent que le temps global de migration varie de quelques secondes à quelques minutes, quant la taille des données de test augmente jusqu'à 100.000 instances **DS3**. Pour l'échantillon le plus volumineux **DS4** contenant 1.000.000 instances, le temps de migration le plus mauvais (*de l'ordre de* $1h30$) est observé pour le pattern *Strict (**PM1**)*. Ce dernier exige des parcours exhaustifs et des comparaisons exactes entre les différents chemins des deux protocoles. Le pattern *Ré-ordonnancement d'activités (**PM2**)* vient en seconde position en termes de temps de migration. Cette observation est justifiée par la nature de ce pattern qui manipule l'opérateur shuffle sur les séquences d'activités. (*calculer toutes les combinaisons possibles et tester l'appartenance d'une trace à cet ensemble*). Pour les autres patterns, le temps d'exécution nécessaire pour faire migrer 1.000.000 d'instances avoisine les 30 minutes, et il descend jusqu'à 15 minutes pour le pattern Extension (**PM5**).

Dans une dernière expérience, nous avons examiné la pertinence des patterns proposés, en calculant les taux des instances migrables : (instances migrables/ instances candidates) pour chaque pattern et en fonction des échantillons de test. Cette expérience a été menée sous l'hypothèse qu'un taux de migration faible peut compromettre toute l'approche proposée. Les résultats obtenus, tels que illustrés dans la figure 7.12, font ressortir que ce taux est autour de 80 % pour les trois patterns **(PM1, PM2** et **PM4)** et il est presque à 100 % pour les deux patterns restants **(PM3** et **PM5)**.

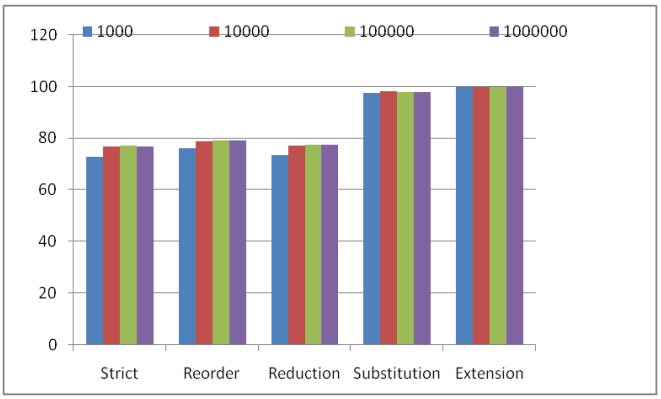

FIGURE 7.12 – Comparaison des taux des instances migrables des différents patterns

Les résultats obtenus sont très prometteurs et consolident les spécifications des patterns identifiés dans notre approche. Néanmoins, les techniques d'adaptateurs [143, 144, 145] demeurent incontournables pour faire migrer les instances restantes et qui ne peuvent être transférées par notre approche.

7.5 Évaluation de la complétude de l'approche

Dans cette section, nous analysons la capacité des patterns de migration proposés à capturer les principales stratégies de migration existantes et à prendre en charge les critères de conformité proposés dans la littérature.

7.5.1 Capture des stratégies de migration existantes

Comme il a été discuté dans la section 4.3, dans le domaine des services web, les notions de *compatibilité historique* et de *compatibilité future* [24, 25, 21, 136, 97] ont été utilisées comme base principale pour définir diverses stratégies de migration des instances actives. Dans notre approche, *la compatibilité historique* peut être accomplie en employant le pattern de migration Strict (**PM1**), comme suit :

$$(\delta_e = v, q_s) \xrightarrow{\text{Strict } ((\delta,q_s),(\beta,q'_t))} (\beta_e = v', q'_t).$$

Ce pattern permet de convertir une instance active i de l'ancien protocole, avec un chemin d'exécution δ en une instance du nouveau protocole, s'il existe un état q'_t qui peut être atteint dans le nouveau protocole en utilisant le chemin d'exécution δ. Par conséquent, ce modèle assure la compatibilité historique des instances migrées.

Pour la *compatibilité future*, le pattern de migration pour la substitution de sous-protocole Sous-P (**PF3**) peut être exploité comme suit :

$$(\delta_e, q_s) \xrightarrow{\text{Sous-P}((\delta,q_s),(\beta,q'_t),(S_P:S_{P'}))} (\beta_e, q'_t)$$

Comme discuté précédemment (cf., section 6.3.2), ce pattern assure qu'une instance i de l'ancien protocole est migrable vers le nouveau protocole, si et seulement si, il

existe un état correspondant dans le nouveau protocole qui peut reproduire tous les comportements qui ont été offerts à i à partir de son état actuel dans l'ancien protocole.

A signaler que la conjonction des deux patterns précédents conduit à une stratégie de migration qui assure à la fois *la compatibilité historique* et *la compatibilité future* des instances migrées. Donc, de réaliser une *compatibilité complète*.

D'autres classes de compatibilité, *basées sur les protocoles clients* et qui sont utiles quand les informations sur les protocoles clients sont disponibles, ont été étudiées dans [24, 135]. Les patterns proposés peuvent être étendus sans difficultés majeures pour répondre à ces classes supplémentaires de compatibilité.

Nous tenons à signaler le fait que la spécification des stratégies de migration qui assurent les cas de compatibilité *historique* et/ou *future* est plus aisée en utilisant nos patterns de migration. Cela est dû au fait que les patterns que nous avons proposé sont *déclaratifs*, dans le sens où ils précisent uniquement **quelles** sont les propriétés à préserver lors de la migration des instances. Par contre, les approches basées sur les modèles de migration discutés dans le chapitre 4 (*section 4.3*) incorporent et intègrent, en même temps, dans la définition de la stratégie de migration à appliquer **quelles** sont les propriétés à préserver et **comment** procéder pour les préserver.

7.5.2 Prise en charge des critères de conformité

Dans le domaine des systèmes workflows et les systèmes de gestion de processus métier (**BPM**), certains critères de conformité manipulés par ces systèmes sont similaires à ceux utilisés dans le domaine des services web, mais ils se manifestent dans un contexte technique différent. Par exemple, la notion de conformité par rapport à l'héritage (*conformité under inheritance*) [23] est très proche de la notion de compatibilité complète, dans le sens où elle est basée sur une relation globale de raffinement entre ancien et nouveau protocoles (*i.e., la relation d'héritage*) qui assure la compatibilité historique et la compatibilité future de toutes les instances de l'ancien processus métiers avec le nouveau. D'autres approches sont basées sur la préservation de l'historique (*préfixe de schéma, états sûrs, la conformité restrictive, critères complets de conformité [23]*). En même temps, une grande partie de ces approches considère les exécutions futures (*séquence de déclenchement de pré-changement : pre-change firing sequence*), [23]). La majorité de ces critères peut être capturée par nos patterns de migration, tel que discuté précédemment.

Deux critères de conformité supplémentaires abordés dans [23] permettent d'étendre la compatibilité historique à deux types particuliers de compatibilité : la compatibilité *insensible au ré-ordonnancement* des activités et la compatibilité *tolérante aux boucles*. Comme présenté dans la section 6.3.1, notre pattern de migration Reord (**PM2**) permet la migration des instances actives d'un ancien protocole, si les changements qui les ont affectés concernent leurs historiques d'exécution et touchent uniquement la modification de l'ordre de certaines activités. (cf., exemple 6.3).

La *tolérance aux boucles* permet d'éviter *d'exclure les instances à migrer du fait que les changements respectifs concernent les boucles* [22, 23]. La compatibilité historique est alors étendue à une instance de l'ancien protocole, si sa trace d'exécution ignore toutes les itérations de la boucle, sauf la dernière, pour qu'elle soit compatible avec le nouveau protocole. Un tel critère peut être capturé dans notre approche en utilisant

le pattern de migration de substitution de sous-chemins *Subst* (**PM3**), comme suit :

$$(\delta_e = v_1 v v^* v_2, q_s) \xrightarrow{\text{Subst } ((\delta,q_s),(\beta,q'_t),(\alpha_e:\alpha'_e))} (\beta_e = v_1 v v_2, q'_t)$$

où v, v_1, v_2 sont des variables de chemins.

Ce pattern capture la sémantique de la compatibilité historique avec *tolérance aux boucles*, dans le sens où les changements dans une boucle n'excluent pas les instances de migrer vers la nouvelle version du protocole. Par exemple, si la boucle CIC est supprimée de l'état Carrière Reconstituée du protocole Retraite de la figure 5.1 et si cette boucle est remplacée par une transition unique CIC, alors ce pattern permet de migrer une instance i ayant un chemin d'exécution ID.VI.CC.CIC.CIC.CIC.VE vers le nouveau protocole, avec comme chemin d'exécution témoin ID.VI.CC.CIC.VE.

Dans cette section, nous avons discuté l'expressivité de notre langage déclaratif de transformation d'instances. Nous avons montré que les patterns de migration proposés peuvent être utilisés pour capturer, de manière déclarative, la plupart des critères de conformité utilisés, à la fois dans le cadre de protocoles métiers des services Web et dans les modèles de workflows. Pour rendre l'image plus complète, il convient de mentionner le fait que notre langage de migration est assez puissant pour permettre d'exprimer des stratégies de migration à granularité fine des instances actives, qui ne sont pas pris en charge par les approches existantes dans l'état de l'art. Par exemple, nous ne sommes pas au courant de l'existence de types d'approches de migration qui utilisent des contraintes d'interdiction, ni des approches qui sont en mesure de définir des critères spécifiques de conformité dans l'esprit de notre *pattern de migration pour la substitution de sous-chemins* (**PM3**). En effet, ce dernier pattern peut être vu comme une **généralisation** des notions de *compatibilité historique et future*, dans le sens où il permet à un utilisateur de définir quand deux instances sont ou ne sont pas compatibles, en fonction du contexte spécifique de son application.

7.6 Conclusion

Dans ce chapitre, nous avons exposé un prototype implémentant notre approche déclarative de migration d'instances actives. L'outil logiciel réalisé répond d'une manière conséquente aux objectifs fixés, à savoir : (i) spécification des protocoles de service par des automates d'état finis déterministes, (ii) simulation des exécutions des services et génération des traces d'exécution, (iii) application des changements sur les protocoles et spécification des patterns de migration gouvernant l'évolution, (iv) analyse de l'impact et migration des instances générées conformément aux patterns proposés.

Il est important de noter que vue les difficultés rencontrées pour l'obtention de données réelles (*journaux logs*), nous étions contraints de procéder aux expérimentations par des simulations de jeux de test générés par notre prototype. Les premiers résultats obtenus sont très satisfaisants en termes de performances.

Pour évaluer l'expressivité et la complétude de notre approche, nous avons montré que les patterns de migration proposés ont la capacité suffisante pour pouvoir prendre en charge, d'une manière plus efficace, les approches de migrations les plus connues dans le domaine des services web et qu'ils vérifient les critères de conformité les plus utilisés dans la littérature relative au domaine.

Chapitre 8

Conclusion Générale

"Rien n'est permanent, sauf le changement"[1]. Effectivement, le changement est une vérité intrinsèque de notre monde physique, où tout phénomène est mouvant et assujetti aux facteurs de son environnement. Cette loi de la "*non-permanence*" ou du changement perpétuel affecte tous les systèmes.

Dans le contexte socio-économique, les changements et les évolutions caractérisent les processus métiers des entreprises qui vivent dans des écosystèmes de plus en plus versatiles. Ces changements exigent, entre autre, qu'un processus métier doit être continuellement actualisé et amélioré. Par conséquent, les organisations doivent s'assurer, à tout moment, que c'est le ***bon protocole métier*** qui est opérationnel. Cette exigence stratégique et tactique influe sur les instances en cours d'exécution et pose, alors, le problème de leur migration vers la nouvelle version du processus métier.

D'autre part, la technologie des services web constitue aujourd'hui une technologie de choix pour l'implémentation des processus métiers trans-organisationnels. Par conséquent, la gestion de l'évolution des protocoles de ces services dans leur phase opérationnelle (*où certains instances sont en cours*) est un problème complexe qui affecte plusieurs dimensions de la sphère de l'organisation.

Dans cette thèse, nous avons apporté une contribution au problème de l'analyse de l'impact des changements. L'approche de migration proposée pour la gestion de l'évolution dynamique des protocoles des services web est déclarative et elle est fondée sur la transformation des instances actives.

8.1 Synthèse des contributions

Nous avons abordé le problème de la migration des instances actives des processus métiers implémentés en tant que services web, suite à l'évolution dynamique des spécifications de ces services. Un cadre formel pour l'analyse de l'impact des changements des protocoles des services web a été formalisé, illustré, implémenté et expérimenté.

Les contributions majeures de la thèse sont les suivantes :

1. Formalisation d'un langage de migration des instances actives basé sur les expressions de chemins et sur les règles de migration.
2. Les patterns les plus fréquents ont été identifiés, formalisés et illustrés par des scénarios réels. Ces patterns étendent le langage proposé et permettent la spécification de différentes stratégies de migration sélectives.
3. Dans l'approche proposée, nous avons traité le problème de la compatibilité stricte des protocoles après l'application des changements. En plus, nous avons proposé de nouvelles classes de compatibilité plus flexibles.

1. Héraclite d'Ephèse. Philosophe pré-socratique, considéré par Hegel comme le père de la pensée dialectique.

4. Au delà de l'examen classique de la gestion des traces historiques, les interactions futures et les scénarios interdits ont été traités et formalisés.
5. La description formelle des patterns de migration constitue un cadre formel solide qui permet d'effectuer des raisonnements sur les connaissances représentant aussi bien, les spécifications du protocole que les traces des exécutions. Par conséquent, nous avons élaboré une technique de composition des patterns de base qui permet la spécification de stratégies de migration plus complexes et qui répondent à des besoins particuliers de gestion.
6. L'approche proposée a été mise en œuvre par un prototype logiciel. L'outil développé offre diverses fonctionnalités très utiles aux gestionnaires de protocole. Le noyau du système gère l'analyse de l'impact du changement sur les instances en cours et fournit les informations de migration adéquates qui sont associées aux patterns de migration prédéfinis.
7. L'approche a été expérimentée sur des données synthétiques. Les résultats obtenus montrent que le système passe à l'échelle et que ses performances sont encourageantes.
8. En termes d'évaluation formelle, le cadre proposé permet de capturer d'une manière plus efficace, les différentes classes de compatibilités existantes et de prendre en charge les critères de conformité utilisés dans la littérature du domaine (*Workflows, processus métiers et services web*)

8.2 Perspectives

Comme directions de recherche futures, nous envisageons de :

1. Consolider le cadre théorique proposé par une étude plus approfondie du langage de migration d'instances et ses propriétés théoriques (*e.g., expressivité vs. complexité*),
2. Approfondir l'étude de la composition des patterns de base pour l'élaboration de stratégies de migration plus complexes,
3. Proposer et développer un modèle formel de coûts qui peut servir de base pour la comparaison de différentes stratégies de migration afin de pouvoir sélectionner la stratégie la plus optimale.
4. Aborder la question du passage à l'échelle par la mise en œuvre de notre approche en utilisant les nouveaux paradigmes de calcul parallèle et distribué (*e.g., MapReduce, Spark*).
5. En fin, l'extension de notre approche au-delà des protocoles métiers pour gérer les modèles de processus plus expressifs (*par exemple, intégrant des activités parallèles, des branchements complexes, les dimensions temps et données, ...*), constitue une direction de recherche future très stimulante.

Bibliographie

[1] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services : Concepts, Architectures and Applications*. Springer, Berlin, 2004.

[2] Chantale Morley, Jean Hughes, Bernard Leblanc, and Olivier Hugues. *Processus métiers et S.I : évaluation, modélisation, mise en oeuvre*. Edition DUNOD, 2007.

[3] Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Mathias Weske. Business process management : A survey. In *Proceedings of the 2003 International Conference on Business Process Management*, BPM'03, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[4] Michael Bell. *Service-Oriented Modeling (SOA) : Service Analysis, Design, and Architecture*. Wiley, 2008.

[5] Thomas Erl. *Service-Oriented Architecture : Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[6] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web services description language (wsdl) version 2.0 part 1 : Core language, June 2007.

[7] W3C. Soap version 1.2 part 0 : Primer (second edition). online, April 2007. W3C Recommendation.

[8] Uddi spec technical committee draft 3.0.2. Oasis committee draft, 2004.

[9] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *Int. J. Web Grid Serv.*, 1(1) :1–30, August 2005.

[10] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Web service conversation modeling : A cornerstone for e-business automation. *IEEE Internet Computing*, 8(1) :46–54, 2004.

[11] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Representing, analysing and managing web service protocols. *Data Knowl. Eng.*, 58(3) :327–357, 2006.

[12] Walid Fdhila, Conrad Indiono, Stefanie Rinderle-Ma, and Manfred Reichert. Dealing with change in process choreographies : Design and implementation of propagation algorithms. *Information Systems*, 49(0) :1 – 24, 2015.

[13] Xiaohui Zhao and Chengfei Liu. Version management for business process schema evolution. *Information Systems*, 38(8) :1046 – 1069, 2013.

[14] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data Knowl. Eng.*, 66(3) :438–466, 2008.

[15] Gregor Joeris and Otthein Herzog. Managing evolving workflow specifications. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems*, COOPIS '98, pages 310–321, Washington, DC, USA, 1998. IEEE Computer Society.

[16] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. *Data Knowl. Eng.*, 24(3) :211–238, January 1998.

[17] W. M. P. VAN DER Aalst. Generic workflow models : How to handle dynamic change and capture management information ? In *Proceedings of the Fourth IECIS International Conference on Cooperative Information Systems*, COOPIS '99, pages 115–, Washington, DC, USA, 1999. IEEE Computer Society.

[18] W. M. P. Van Der Aalst. Exterminating the dynamic change bug : A concrete approach to support workflow change. *Information Systems Frontiers*, 3(3) :297–317, September 2001.

[19] Paulo Dias, Pedro Vieira, and António Rito-Silva. Dynamic evolution in workflow management systems. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, DEXA '03, pages 254–, Washington, DC, USA, 2003. IEEE Computer Society.

[20] SHAZIA W. SADIQ. Workflows in dynamic environments - can they be managed ? In *CODAS*, pages 165–176, 1999.

[21] Nannette Liske, Niels Lohmann, Christian Stahl, and Karsten Wolf. Another approach to service instance migration. In *ICSOC-ServiceWave '09*, pages 607–621, Berlin, Heidelberg, 2009. Springer-Verlag.

[22] Manfred Reichert, Stefanie Rinderle, and Peter Dadam. On the common support of workflow type and instance changes under correctness constraints. In *CooplS*, number 2888 in LNCS, pages 407–425. Springer, 2003.

[23] Stefanie Rinderle, Manfred Reichert, and Peter Dadam. Correctness criteria for dynamic changes in workflow systems : A survey. *Data Knowl. Eng.*, 50(1) :9–34, July 2004.

[24] Seung Hwan Ryu, Fabio Casati, Halvard Skogsrud, Boualem Benatallah, and Régis Saint-Paul. Supporting the dynamic evolution of web service protocols in service-oriented architectures. *ACM Trans. Web*, 2(2) :1–46, May 2008.

[25] Seung Hwan Ryu, Régis Saint-Paul, Boualem Benatallah, and Fabio Casati. A framework for managing the evolution of business protocols in web services. In *Proceedings of the fourth Asia-Pacific conference on Comceptual modelling - Volume 67*, APCCM '07, pages 49–59, Ballarat, Australia, 2007. Australian Computer Society, Inc.

[26] Frédéric Cuppens, Christophe Garion, Guillaume Piolle, and Nora Cuppens-Boulahia. Normes et logique déontique. In Pierre Marquis, Odile Papini, and Henri Prade, editors, *Panorama de l'Intelligence Artificielle : Volume 1. Représentation des connaissances et formalisation des raisonnements*, pages 215–237. Cépaduès Editions, 2014. ISBN : 9782364930414.

[27] Angel Lagares Lemos, Florian Daniel, and Boualem Benatallah. Web service composition : A survey of techniques and tools. *ACM Comput. Surv.*, 48(3) :33 :1–33 :41, December 2015.

[28] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella, and Fabio Patrizi. Automatic service composition and synthesis : the roman model. *IEEE Data Eng. Bull.*, 31(3) :18–22, 2008.

[29] Ingo Weber, Hye-Young Paik, and Boualem Benatallah. Form-based web service composition for domain experts. *ACM Trans. Web*, 8(1) :2 :1–2 :40, December 2013.

[30] Kadima Hubert and Monfort Valérie. *Les Web Services techniques, démarches et outils XML, WSDL, SOAP, UDDI, Rosetta, UML.* DUNOD, 2003.

[31] W3C Working Group. Web services architecture, 11 february 2004.

[32] Rudi Studer, Stephan Grimm, and Andreas Abecker. *Semantic Web Services : Concepts, Technologies, and Applications.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[33] Jabu Mtsweni, Elmarie Biermann, and Laurette Pretorius. isemserv : Towards the engineering of intelligent semantic-based services. In *Proceedings of the 10th International Conference on Current Trends in Web Engineering*, ICWE'10, pages 550–559, Berlin, Heidelberg, 2010. Springer-Verlag.

[34] Mohamed Sellami, Samir Tata, Zakaria Maamar, and Bruno Defude. A recommender system for web services discovery in a distributed registry environment. In *Fourth International Conference on Internet and Web Applications and Services, ICIW 2009, 24-28 May 2009, Venice/Mestre, Italy*, pages 418–423, 2009.

[35] Nomane Ould Ahmed M'Bareck, Samir Tata, and Zakaria Maamar. Towards an approach for enhancing web services discovery. In *16th IEEE International Workshops on Enabling Technologies : Infrastructures for Collaborative Enterprises (WETICE 2007), 18-20 June 2007, Paris, France*, pages 357–364, 2007.

[36] Erbin Lim, Philippe Thiran, Zakaria Maamar, and Jamal Bentahar. On the analysis of satisfaction for web services selection. In *2012 IEEE Ninth International Conference on Services Computing, Honolulu, HI, USA, June 24-29, 2012*, pages 122–129, 2012.

[37] Silvia Calegari, Marco Comerio, Andrea Maurino, Emanuele Panzeri, and Gabriella Pasi. A semantic and information retrieval based approach to service contract selection. In *Service-Oriented Computing - 9th International Conference, ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings*, pages 389–403, 2011.

[38] Hongbing Wang, Shizhi Shao, Xuan Zhou, Cheng Wan, and Athman Bouguettaya. Web service selection with incomplete or inconsistent user preferences. In *Service-Oriented Computing, 7th International Joint Conference, ICSOC-ServiceWave 2009, Stockholm, Sweden, November 24-27, 2009. Proceedings*, pages 83–98, 2009.

[39] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *TWEB*, 1(1), 2007.

[40] Mohammad Alrifai, Thomas Risse, Peter Dolog, and Wolfgang Nejdl. A scalable approach for qos-based web service selection. In George Feuerlicht and Winfried Lamersdorf, editors, *Service-Oriented Computing – ICSOC 2008 Workshops*, volume 5472 of *Lecture Notes in Computer Science*, pages 190–199. Springer Berlin Heidelberg, 2009.

[41] W3C. Extensible markup language (xml) 1.0 (fifth edition) : W3c recommendation 26 november 2008. http ://www.w3.org/TR/xml/.

[42] Harvey Bingham. Sgml syntax summary. http ://www.oasis-open.org/cover/sgmlsyn/contents.htm/.

[43] W3C. On sgml and html (in the html 4.01 specification). http ://www.w3.org/TR/html401/intro/sgmltut.html/.

[44] Charles F. Goldfarb. The roots of sgml – a personal recollection.

[45] Arnaud LeJacobs Ian Raggett, Dave Hors. Html 4.01 specification. http ://www.w3.org/TR/html401/, December 1999.

[46] Html 5.1, w3c working draft. http ://www.w3.org/TR/2015/WD-html51-20151008/, 2015.

[47] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic service composition based on behavioral descriptions. *Int. J. Cooperative Inf. Syst.*, 14(4) :333–376, 2005.

[48] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *Proceedings of the First International Conference on Semantic Web Services and Web Process Composition*, SWSWPC'04, pages 43–54, Berlin, Heidelberg, 2005. Springer-Verlag.

[49] Ramy Ragab HASSEN. *Automatic composition of protocol-based web services.* PhD thesis, Clermont Ferrand University -France-, July 2009.

[50] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic composition of e-services that export their behavior. In *ICSOC*, pages 43–58, Dec. 2003.

[51] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 77–88, New York, NY, USA, 2002. ACM.

[52] Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference - Volume 17*, ADC '03, pages 191–200, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[53] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. *Artificial Intelligence : Methodology, Systems, and Applications : 11th International Conference, AIMSA 2004, Varna, Bulgaria, September 2-4, 2004. Proceedings*, chapter Planning and Monitoring Web Service Composition, pages 106–115. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[54] OASIS. Web services business process execution language version 2.0. http ://docs.oasis-open.org/wsbpel/2.0/, 2007.

[55] Ramy Ragab Hassen, Farouk Toumani, and Lhouari Nourine. Web services composition is decidable. In *11th International Workshop on the Web and Databases, WebDB 2008, Vancouver, BC, Canada, June 13, 2008*, 2008.

[56] Keita Fujii and Tatsuya Suda. Semantics-based context-aware dynamic service composition. *ACM Trans. Auton. Adapt. Syst.*, 4(2) :12 :1–12 :31, May 2009.

[57] M. P. Papazoglou and D. Georgakopoulos. Introduction : Service-oriented computing. *Commun. ACM*, 46(10) :24–28, October 2003.

[58] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing : State of the art and research challenges. *Computer*, 40(11) :38–45, November 2007.

[59] Mike P. Papazoglou and Willem-Jan Heuvel. Service oriented architectures : Approaches, technologies and research issues. *The VLDB Journal*, 16(3) :389–415, July 2007.

[60] Karim Baïna, Boualem Benatallah, Fabio Casati, and Farouk Toumani. Model-driven web service development. In Anne Persson and Janis Stirna, editors, *Advanced Information Systems Engineering, 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Proceedings*, volume 3084 of *Lecture Notes in Computer Science*, pages 290–306. Springer, 2004.

[61] Julien Ponge, Boualem Benatallah, Fabio Casati, and Farouk Toumani. Fine-grained compatibility and replaceability analysis of timed web service protocols. In *ER*, pages 599–614, 2007.

[62] Ali Khebizi. External behavior modeling enrichment of web services by transactional constraints. In *ICSOC PhD Symposium*, 2008.

[63] RosettaNet. Rosettanet pip directory. http ://www.rosettanet.org/.

[64] Khebizi Ali, Seridi-Bouchelaghem Hassina, Chemakhi Imed, and Hychem Bekakria. Service substitution analysis in protocols evolution context. In *ICWIT*, pages 12–21, 2012.

[65] AXEL MARTENS. On Compatibility of Web Services. *Petri Net Newsletter*, 65 :12–20, 2003.

[66] Nick Russell, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Petia Wohed. On the suitability of uml 2.0 activity diagrams for business process modelling. In *Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling - Volume 53*, APCCM '06, pages 95–104, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[67] Object Management Group. Business process modeling notation (bpmn) version 1.0. omg final adopted specification. object management group, 2006.

[68] Unified modeling language (omg uml), infrastructure, v2.1.2. http ://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/.

[69] Charles Lanny Forgy. *On the efficient implementation of production systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1979.

[70] Web services security. https ://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

[71] The platform for privacy preferences 1.0 (p3p1.0) specification. http ://www.w3.org/TR/2002/REC-P3P-20020416/.

[72] Ws-trust 1.3. http ://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.doc.

[73] Ws-secureconversation 1.3. http ://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3-os.doc.

[74] Web services policy 1.5 - framework. http ://www.w3.org/TR/2007/REC-ws-policy-20070904.

[75] Web services federation language (wsfederation). http ://specs.xmlsoap.org/ws/2006/12/federation/ws-federation.pdf.

[76] Frank Leymann. Web Services Flow Language (WSFL 1.0). Technical report, IBM, May 2001.

[77] Web service choreography interface (wsci) 1.0. http ://www.w3.org/TR/2002/NOTE-wsci-20020808.

[78] Web services conversation language (wscl) 1.0. http ://www.w3.org/TR/2002/NOTE-wscl10-20020314/.

[79] Paolo Traverso and Marco Pistore. Automated composition of semantic web services into executable processes. pages 380–394. 2004.

[80] Rik Eshuis, Freddy Lécué, and Nikolay Mehandjiev. Flexible construction of executable service compositions from reusable semantic knowledge. *ACM Trans. Web*, 10(1) :5 :1–5 :27, February 2016.

[81] Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. Composing web services on the semantic web. *The VLDB Journal*, 12(4) :333–351, November 2003.

[82] Gunwoo Kim and Yongmoo Suhh. Ontology-based semantic matching for business process management. *SIGMIS Database*, 41(4) :98–118, November 2010.

[83] Kyriakos Kritikos and Dimitris Plexousakis. Requirements for qos-based web service description and discovery. *IEEE Trans. Serv. Comput.*, 2(4) :320–337, October 2009.

[84] Srividya Kona, Ajay Bansal, M. Brian Blake, Steffen Bleul, and Thomas Weise. Wsc-2009 : A quality of service-oriented web services challenge. In *Proceedings of the 2009 IEEE Conference on Commerce and Enterprise Computing*, CEC '09, pages 487–490, Washington, DC, USA, 2009. IEEE Computer Society.

[85] Hyunyoung Kil, Reeseo Cha, and Wonhong Nam. Transaction history-based web service composition for uncertain qos. *Int. J. Web Grid Serv.*, 12(1) :42–62, January 2016.

[86] A.Blum. Uddi as an extended web services registry : Versioning, quality of service, and more. *White Paper, SOA*, 4(6), 2004.

[87] Preeti Marwaha, Hema Banati, and Punam Bedi. {UDDI} extensions for temporally customized web services. *Procedia Technology*, 10 :184 – 190, 2013. First International Conference on Computational Intelligence : Modeling Techniques and Applications (CIMTA) 2013.

[88] Asim El Sheikh, Alia Sabri, Bassam Al Shargabi, and Osama Al-haj Hassan. Quality-of-service based web service composition and execution framework. *Int. J. Inf. Technol. Web Eng.*, 6(3) :57–74, July 2011.

[89] Rui Wang, Sumedha Ganjoo, John A. Miller, and Eileen T. Kraemer. Ranking-based suggestion algorithms for semantic web service composition. In *Proceedings of the 2010 6th World Congress on Services*, SERVICES '10, pages 606–613, Washington, DC, USA, 2010. IEEE Computer Society.

[90] Chun Ouyang, Marlon Dumas, Arthur H.M. ter Hofstede, and Wil M.P. van der Aalst. Pattern-based translation of bpmn process models to bpel web services. *International Journal of Web Services Research (JWSR)*, 5(1) :42–62, 2007.

[91] Andreas Wombacher, Peter Frankhauser, and Erich Neuhold. Transforming BPEL into annotated Deterministic Finite State Automata enabling process annotated service discovery. In *2nd Intl. Conference on Web Services (ICWS 04)*, pages 316–323, San Diego, California, USA, July 2004. IEEE.

[92] Gerti Kappel, Philip Langer, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. Conceptual modelling and its theoretical foundations. chapter Model Transformation By-example : A Survey of the First Wave, pages 197–215. Springer-Verlag, Berlin, Heidelberg, 2012.

[93] Franck Cassez and Olivier-H. Roux. Structural translation from time petri nets to timed automata. *Electronic Notes in Theoretical Computer Science*, 128(6) :145 – 160, 2005. Proceedings of the Fouth International Workshop on Automated Verification of Critical Systems (AVoCS 2004)Automated Verification of Critical Systems 2004.

[94] Xi Wang, Huaikou Miao, and Liang Guo. Towards automatic transformation from uml model to fsm model for web applications. *JSEA*, 1(1) :68–75, 2008.

[95] Kais Klai, Samir Tata, and Jörg Desel. Symbolic abstraction and deadlock-freeness verification of inter-enterprise processes. In *BPM*, pages 294–309, 2009.

[96] Yi Wang and Ying Wang. A survey of change management in service-based environments. *Serv. Oriented Comput. Appl.*, 7(4) :259–273, December 2013.

[97] Mike P. PAPAZOUGLOU. The challenges of service evolution. In *Proceedings of the 20th international conference on Advanced Information Systems Engineering*, CAiSE '08, pages 1–15, Berlin, Heidelberg, 2008. Springer-Verlag.

[98] Marlon Dumas, Murray Spork, and Kenneth Wang. Adapt or perish : Algebra and visual notation for service interface adaptation. In *Proceedings of the 4th International Conference on Business Process Management*, BPM'06, pages 65–80, Berlin, Heidelberg, 2006. Springer-Verlag.

[99] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition.* Springer, 2012.

[100] Catriel Beeri, Anat Eyal, Tova Milo, and Alon Pilberg. Monitoring business processes with queries. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 603–614. VLDB Endowment, 2007.

[101] Oussama Mohammed Kherbouche. *Contribution à la gestion de l'évolution des processus métiers.* PhD thesis, Université du Littoral Côte d'Opale, -France-, 2013.

[102] Minor Mirjam, Schmalen Daniel, Koldehoff Andreas, and Bergmann Ralph. Structural adaptation of workflows supported by a suspension mechanism stand by case-based reasoning. In *16th IEEE International Workshops on Enabling Technologies : Infrastructures for Collaborative Enterprises (WETICE 2007), 18-20 June 2007, Paris, France*, pages 370–375, 2007.

[103] Selmin Nurcan. A survey on the flexibility requirements related to business processes and modeling artifacts. In *Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, HICSS '08, pages 378–, Washington, DC, USA, 2008. IEEE Computer Society.

[104] Wil M. P. van der Aalst. Challenges in business process analysis. In *Enterprise Information Systems, 9th International Conference, ICEIS, 2007, Funchal, Madeira, June 12-16, 2007, Revised Selected Papers*, pages 27–42, 2007.

[105] Gil Regev, Pnina Soffer, and Rainer Schmidt. Taxonomy of flexibility in business processes. In Gil Regev, Pnina Soffer, and Rainer Schmidt, editors, *BPMDS*, volume 236 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[106] Frederick M. Haney. Module connection analysis : A tool for scheduling software debugging activities. In *Proceedings of the December 5-7, 1972, Fall Joint Computer Conference, Part I*, AFIPS '72 (Fall, part I), pages 173–179, New York, NY, USA, 1972. ACM.

[107] S. L. Pfleeger and S. A. Bohner. A framework for software maintenance metrics. In *The IEEE Transactions on Software Engineering*, pages 320–327, San Diego CA, USA, 1990. IEEE Computer Society.

[108] Robert S. Arnold. *Software Change Impact Analysis*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[109] Gary Hamel and Liisa Välikangas. The quest for resilience. *Harvard Business Review*, (September) :52–63, 2003.

[110] Nikos C. Tsourveloudis and Kimon P. Valavanis. On the measurement of enterprise agility. *Journal of Intelligent and Robotic Systems*, pages 329–342, 2002.

[111] Barbara Dellen, Frank Maurer, and Gerhard Pews. Knowledge-based techniques to increase the flexibility of workflow management. *Data Knowl. Eng.*, 23(3) :269–295, September 1997.

[112] Khalid Belhajjame, Genoveva Vargas-Solar, and Christine Collet. Towards an adaptable workflow management system. In Noureddine Mouaddib, editor, *17èmes Journées Bases de Données Avancées, BDA 2001, 29 octobre - 2 novembre, Agadir (Maroc), Actes (Informal Proceedings).*, 2001.

[113] Justus Klingemann. Controlled flexibility in workflow management. In *Advanced Information Systems Engineering, 12th International Conference CAiSE 2000, Stockholm, Sweden, June 5-9, 2000, Proceedings*, pages 126–141, 2000.

[114] Markus Kradolfer and Andreas Geppert. Dynamic workflow schema evolution based on workflow type versioning and workflow migration. In *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems, Edinburgh, Scotland, September 2-4, 1999*, pages 104–114, 1999.

[115] Mathias Weske. Flexible modeling and execution of workflow activities. In *Thirty-First Annual Hawaii International Conference on System Sciences, Kohala Coast, Hawaii, USA, January 6-9, 1998*, pages 713–722, 1998.

[116] Alessandra Agostini and Giorgio De Michelis. A light workflow management system using simple process models. *Computer Supported Cooperative Work*, 9(3/4) :335–363, 2000.

[117] Mathias Weske. Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In *34th Annual Hawaii International Conference on System Sciences (HICSS-34), January 3-6, 2001, Maui, Hawaii, USA*, 2001.

[118] M. Reichert and P. Dadam. Adeptflex-supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, pages 93–129, 1998.

[119] Mohamed Ahmed-Nacer and Jacky Estublier. Schema evolution in software engineering databases - a new approach in adele environment. *Computers and Artificial Intelligence*, 19(2) :183–203, 2000.

[120] José Andany, Michel Léonard, and Carole Palisser. Management of schema evolution in databases. In *Proceedings of the 17th International Conference on Very Large Data Bases*, VLDB '91, pages 161–170, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[121] Peter McBrien and Alexandra Poulovassilis. Schema evolution in heterogeneous database architectures, a schema transformation approach. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, CAiSE '02, pages 484–499, London, UK, UK, 2002. Springer-Verlag.

[122] Natalya F. Noy and Michel Klein. Ontology evolution : Not the same as schema evolution. *Knowl. Inf. Syst.*, 6(4) :428–440, July 2004.

[123] Michel C. A. Klein, Atanas Kiryakov, Damyan Ognyanov, and Dieter Fensel. Finding and characterizing changes in ontologies. In *Proceedings of the 21st International Conference on Conceptual Modeling*, ER '02, pages 79–89, London, UK, UK, 2002. Springer-Verlag.

[124] Longfei Jin, Lei Liu, and Dong Yang. A model transformation based conceptual framework for ontology evolution. In *Proceedings of the 9th international conference on Knowledge-Based Intelligent Information and Engineering Systems - Volume Part I*, KES'05, pages 325–331, Berlin, Heidelberg, 2005. Springer-Verlag.

[125] Edgar Jembere, Sibusiso S. Xulu, and Matthew O. Adigun. Automatic ontology evolution in open and dynamic computing environments. In *Proceedings of the Second international conference on Computational collective intelligence : technologies and applications - Volume Part III*, ICCCI'10, pages 122–132, Berlin, Heidelberg, 2010. Springer-Verlag.

[126] Alexander Stuckenholz. Component evolution and versioning state of the art. *SIGSOFT Softw. Eng. Notes*, 30(1), January 2005.

[127] Andreas RAUSCH. Software evolution in componentware using requirements/assurances contracts. In *Proceedings of the 22nd international conference on Software engineering*, ICSE '00, pages 147–156, New York, NY, USA, 2000. ACM.

[128] Bennett P. Lientz and E. Burton Swanson. *Software Maintenance Management.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1980.

[129] Tom Mens and Tom Tourwé. A survey of software refactoring. *IEEE Trans. Softw. Eng.*, 30(2) :126–139, February 2004.

[130] Martin FOWLER. *Refactoring : Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA, 1999.

[131] Paolo Bottoni, Francesco Parisi-Presicce, and Gabriele Taentzer. Specifying integrated refactoring with distributed graph transformations. In *AGTIVE*, pages 220–235, 2003.

[132] Jean-Luc Hainaut, Anthony Cleve, Jean Henrard, and Jean-Marc Hick. Migration of legacy information systems. In *Software Evolution*, pages 105–138. 2008.

[133] Philippe Thiran, Jean-Luc Hainaut, Geert-Jan Houben, and Djamal Benslimane. Wrapper-based evolution of legacy information systems. *ACM Trans. Softw. Eng. Methodol.*, 15(4) :329–359, October 2006.

[134] Nathan D. Ryan and Alexander L. Wolf. Using event-based translation to support dynamic protocol evolution. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 408–417, Washington, DC, USA, 2004. IEEE Computer Society.

[135] Halvard Skogsrud, Boualem Benatallah, Fabio Casati, and Farouk Toumani. Managing impacts of security protocol changes in service-oriented applications. In *ICSE*, pages 468–477, 2007.

[136] Ahmed Azough, Emmanuel Coquery, and Mohand-Said Hacid. Supporting web service protocol changes by propagation. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '09, pages 438–441, Washington, DC, USA, 2009. IEEE Computer Society.

[137] Wei Hu and Yuzhong Qu. Discovering simple mappings between relational database schemas and ontologies. pages 225–238. 2008.

[138] Hong-Hai Do and Erhard Rahm. Matching large schemas : Approaches and evaluation. *Inf. Syst.*, 32(6) :857–885, September 2007.

[139] L. Bocchi, S. Gorton, and S. Reiff-Marganiec. Engineering service oriented applications : From stpowla processes to SRML models. In *FASE 2008*, pages 163–178, 2008.

[140] Seung Hwan Ryu, Abdelkarim Erradi, Khaled M. Khan, Saleh Alhazbi, and Boualem Benatallah. *Semantics-Based Approach for Dynamic Evolution of Trust Negotiation Protocols in Cloud Collaboration*, pages 518–526. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[141] Piotr Kaminski, Hausi Müller, and Marin Litoiu. A design for adaptive web service evolution. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, SEAMS '06, pages 86–92, New York, NY, USA, 2006. ACM.

[142] Yi Wang and Ying Wang. A survey of change management in service-based environments. *SOCA*, 7(4) :259–273.

[143] Boualem Benatallah, Fabio Casati, Daniela Grigori, Hamid R. Motahari Nezhad, and Farouk Toumani. Developing adapters for web services integration. *Lecture Notes in Computer Science*, 3520 :415–429, 2005. Advanced Information Systems Ingeneering.

[144] Marios Fokaefs and Eleni Stroulia. Wsdarwin : Automatic web service client adaptation. In *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '12, pages 176–191, Riverton, NJ, USA, 2012. IBM Corp.

[145] Ali Aït-Bachir, Marlon Dumas, and Marie-Christine Fauvet. Service-oriented computing — icsoc 2008 workshops. chapter Detecting Behavioural Incompatibilities Between Pairs of Services, pages 79–90. Springer-Verlag, Berlin, Heidelberg, 2009.

[146] Karin Becker, Jim Pruyne, Sharad Singhal, Andre Lopes, and Dejan Milojicic. Automatic determination of compatibility in evolving services. *Int. J. Web Serv. Res.*, 8(1) :21–40, January 2011.

[147] Bruno Vollino and Karin Becker. Usage profiles : A process for discovering usage patterns over web services and its application to service evolution. *Int. J. Web Serv. Res.*, 10(1) :1–28, January 2013.

[148] Yi Wang, Jian Yang, Weiliang Zhao, and Jianwen Su. Change impact analysis in service-based business processes. *Serv. Oriented Comput. Appl.*, 6(2) :131–149, June 2012.

[149] Khubaib Amjad Alam, Rodina Ahmad, Adnan Akhunzada, Mohd Hairul Nizam Md Nasir, and Samee U. Khan. Impact analysis and change propagation in service-oriented enterprises. *Inf. Syst.*, 54(C) :43–73, December 2015.

[150] Ibm software. Technical report, www-01.ibm.com/software.

[151] Oracle Web Services Manager. Fiche technique. Technical report, www.itplace.tv.

[152] www.delawareconsulting.be/index.aspx. Sap netweaver. Technical report.

[153] SAP. Mega solman. In *http ://sapinsider.wispubs.com/Assets/Articles/2009/August/Model-Your-Business-Processes-With-SAP-Solution-Manager*.

[154] Xavier Claude et Nÿco. Bonita bpm community. *linuxfr.org/news/bonita-bpm-community*, 2013.

[155] Processmaker is your open source bpm workflow software solution. Technical report, www.processmaker.com.

[156] Philippe Desfray and Gilbert Raymond. *Modeling Enterprise Architecture with TOGAF : A Practical Guide Using UML and BPMN*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2014.

[157] Daniela Berardi, Fahima Cheikh, Giuseppe De Giacomo, and Fabio Patrizi. Automatic service composition via simulation. *IJFCS*, 19(2) :429–451, 2008.

[158] C.A. Middelburg and M.A. Reniers. Introduction to process theory. Technical report, Technische Universiteit Eindhoven, 2004.

[159] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[160] Nils Erik Flick and Manfred Kudlek. Properties of languages with catenation and shuffle. In *CS-P*, volume 928 of *CEUR Workshop Proceedings*, pages 91–102. CEUR-WS.org, 2012.

[161] Ali Khebizi, Hassina Seridi-Bouchelaghem, Bouallem Benatallah, and Farouk Toumani. A declarative language to support dynamic evolution of web service business protocols. *Service Oriented Computing and Applications*, 11(2) :163–181, 2017.

[162] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange : Semantics and query answering. *Theor. Comput. Sci.*, 336(1) :89–124, May 2005.

[163] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web : From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[164] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[165] Yehia Elshater, Patrick Martin, and Ehab Hassanein. Using design patterns to improve web service performance. In *2015 IEEE International Conference on Services Computing, SCC 2015, New York City, NY, USA, June 27 - July 2, 2015*, pages 746–749, 2015.

[166] Francy D. Rodríguez, Silvia Teresita Acuña, and Natalia Juristo Juzgado. Design and programming patterns for implementing usability functionalities in web applications. *Journal of Systems and Software*, 105 :107–124, 2015.

[167] Mahesh Lal. *Neo4J Graph Data Modeling*. Packt Publishing, 2015.

[168] Onofrio Panzarino. *Learning Cypher*. Packt Publishing, 2014.

[169] Bob DuCharme. *Learning SPARQL*. O'Reilly Media, Inc., 2011.

[170] Soosai Raj Jignesh M. Patel Jing Fan, Adalbert Gerald. The case against specialized graph analytics engines. *CIDR*, 2015.

[171] R. Ciucanu G. H. L. Fletcher A. Lemay G. Bagan, A. Bonifati and N. Advokaat. gmark : Schema-driven generation of graphs and queries. http ://arxiv.org/abs/1511.08386, 2015.

[172] R. Ciucanu G. H. L. Fletcher A. Lemay G. Bagan, A. Bonifati and N. Advokaat. Generating flexible workloads for graph databases. *PVLDB*, 2016.

[173] Sherif Sakr and Ghazi Al-naymat. Efficient relational techniques for processing graph queries, 2010.

[174] Torsten Grust, Manuel Mayr, Jan Rittinger, Sherif Sakr, and Jens Teubner. A sql : 1999 code generator for the pathfinder xquery compiler. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 1162–1164, New York, NY, USA, 2007. ACM.

[175] jdom, api java permettant la manipulation de documents xml. Technical report, http ://www.jdom.org/.

[176] Manipuler les documents xml avec java et jdom. *http ://cyno-ber.developpez.com/tutoriel/java/xml/jdom/.*

[177] Mysql connector/odbc developer guide. *http ://dev.mysql.com/doc/connector-odbc/en/.*

.