Chapitre 2

Modélisation et test de la composition de services Web

Sommaire

2.1 Introduction

Le Web nous a servi pendant plus d'une vingtaine d'année. Il a été conçu pour les interactions entre humains et applications. L'intervention humaine est nécessaire pratiquement tout le temps. Depuis quelques années, des efforts ont été déployés pour utiliser le Web comme une interface machine-à-machine par le biais de la notion de services Web. Ces derniers (services Web) permettent des interactions entre applications de façon systématique et standardisée, et cela dans des environnements hétérogènes.

La possibilité d'intégrer ou de composer des services existants en nouveaux services est la plus importante fonctionnalité assurée par les architectures orientées services [2, 7]. La composition de services doit permettre de créer, d'exécuter, de maintenir des services qui reposent sur d'autres services et ceci de façon efficace. Cette composition de services peut être effectuée par orchestration ou chorégraphie. Elle peut être réalisée en utilisant des langages de programmation (e.g. C++, Java). L'utilisation des langages dédiés à la composition de services offre plus d'avantages dans les processus de développement, d'automatisation, de maintenance et d'intégration des services composés. BPEL [1] s'est imposé, ces dernières années, comme un standard d'orchestration de services Web, et a été accepté par les principaux industriels intervenant dans le domaine des architectures orientée services et des services Web.

Le test est une tâche importante dans le processus de développement d'un logiciel ou d'un système. C'est un moyen de validation qui a pour but de mettre en évidence les défauts du logiciel. De son coté, le test des services Web est devenu un facteur important pour le succès du paradigme des services Web. En raison des propriétés spécifiques et complexes des services Web et de leur composition, les modèles et les techniques de test des systèmes et des logiciels doivent être revisités dans le domaine des services Web.

Le test des services composés décrits en BPEL (ou processus BPEL) nécessite un formalisme adapté pour la modélisation de l'orchestration des services Web. Plus particulièrement, ce formalisme doit être capable de décrire les différentes constructions du langage BPEL et de prendre en compte les différentes particularités de ce langage d'orchestration telles que la gestion des exceptions, la corrélation des messages, la compensation des activités, et la synchronisation et la gestion des liens.

Nous proposons dans ce chapitre une présentation du contexte dans lequel s'inscrivent nos travaux, à savoir, la modélisation et le test de l'orchestration des services Web. Nous introduisons, dans un premier temps, le concept de services Web ainsi que la notion de composition de services Web. Nous abordons, par la suite, le test des logiciels et ses différents types. Ensuite, nous présentons brièvement BPEL qui est le langage standard d'orchestration des services Web. Nous rappelons les principaux modèles proposés pour la spécification de la composition des services Web, plus précisément des processus BPEL, ainsi que les différentes techniques et approches définies pour le test des services Web composés décrits en BPEL.

2.2 Présentation des services Web

Dans cette section, nous présenterons l'architecture orientée services et sa principale réalisation : les services Web. Nous donnons deux définitions des services Web et décrivons brièvement leurs principaux standards et technologies. Enfin, nous abordons le concept de composition des services Web en explicitant les deux méthodes de composition : l'orchestration et la chorégraphie.

2.2.1 Architecture orientée services

Une architecture orientée services, notée SOA [2, 7], est une architecture logicielle visant à mettre en place un système d'information constitué de services applicatifs indépendants et interconnectés. L'architecture SOA permet la réutilisation des applications et des services existants, ainsi de nouveaux services peuvent être créés à partir d'une infrastructure informatique des systèmes déjà existante. En d'autres termes, SOA permet aux entreprises de tirer partie des investissements existants en leur permettant de réutiliser des applications existantes, en leur offrant une interopérabilité entre applications et technologies hétérogènes. L'objectif d'une architecture SOA est de décomposer une fonctionnalité en un ensemble de services et de décrire leurs interactions. Ces services peuvent être utilisés par des processus métier (i.e. services Web composés) ou par des clients dans différentes applications.

Dans une architecture SOA, comme le montre la Figure 2.1, les fournisseurs de services publient (ou enregistrent) leurs services dans un annuaire de services (qui peut être publique ou local à un réseau d'entreprise par exemple). Cet annuaire est utilisé par les utilisateurs (i.e. les clients de services) pour trouver des services vérifiant certains critères ou correspondant à une certaine description. Si l'annuaire contient de tels services, il envoie au client les descriptions de ces services avec un contrat d'utilisation. Le client fait alors son choix, s'adresse au fournisseur et invoque le service Web.

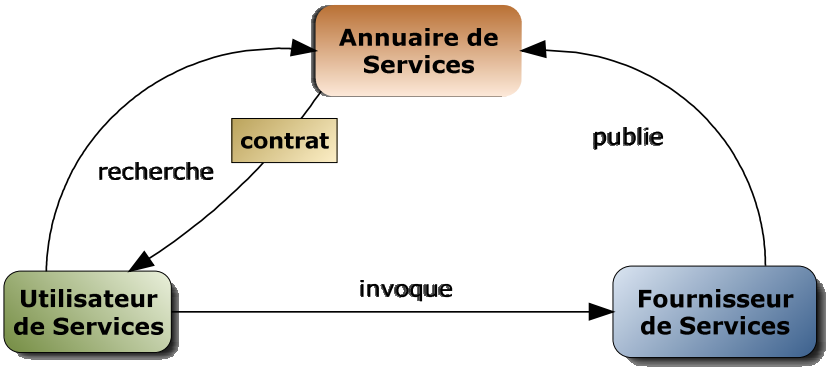

FIGURE 2.1 – Le modèle fonctionnel d'une architecture orientée services (SOA)

Les services Web sont largement utilisés dans le cadre des architectures SOA, étant donné qu'ils peuvent être combinés pour réaliser de nouveaux processus (ou services) plus rapidement qu'un développement traditionnel. Les services Web sont la réalisation (ou l'instance) la plus importante d'une architecture SOA. Ils peuvent faire appel à différents services et peuvent être eux mêmes déployés en tant que nouveaux services.

2.2.2 Services Web

Il existe de nombreuses définitions des services Web. Nous citerons deux d'entre elles. Selon Mark Colan (IBM), les services Web sont : « *des applications modulaires basées sur internet qui exécutent des tâches précises et qui respectent un format spécifique* ».

Nous donnons la définition du W3C [1] pour les services Web, qui est plus complète étant donné qu'elle cite les principaux standards des services Web : « *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.* »

En d'autres mots, les services Web sont des applications auto descriptives, modulaires et faiblement couplées fournissant un modèle simple de programmation et de déploiement d'applications. Ils reposent principalement sur des technologies basées sur XML pour la structure et le contenu de messages échangés entre services (SOAP), pour la description des services (WSDL), pour la découverte des services (UDDI) et pour leurs orchestrations (BPEL). L'ensemble des ces technologies se retrouvent dans l'abréviation WS-*.

Le modèle des services Web, illustré ci-dessous dans la Figure 2.2, instancie celui de l'architecture SOA qui est représenté ci-dessus dans la Figure 2.1.

Dans la suite, nous allons décrire brièvement les principaux standards des services Web : XML, HTTP, WSDL, SOAP et UDDI.

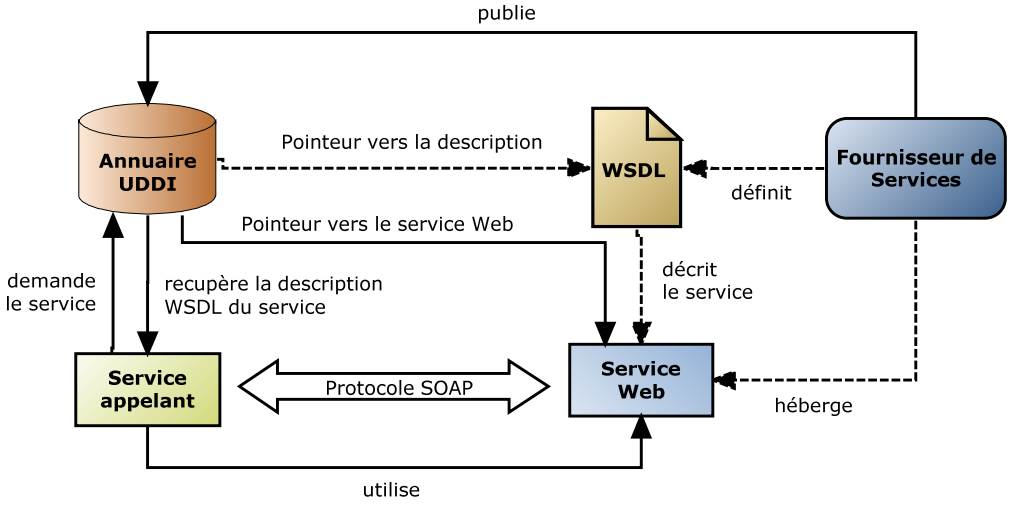

FIGURE 2.2 – Le modèle des services Web

1. World Wide Web Consortium : `http://www.w3.org/`

XML

XML est un langage utilisé pour décrire les messages échangés entre applications [8]. Il permet une représentation textuelle des données. Dans les services Web, c'est plus particulièrement XML schema qui est mis en œuvre [9] et qui permet la description des structures de données en XML.

HTTP

HTTP est un protocole largement utilisé et mis en œuvre pour le transport des formulaires de pages HTML [10]. Dans le cadre des services Web, il a en charge le transport des messages échangés.

WSDL

WSDL est un langage basé sur XML permettant la description unifiée des interfaces (publiques) de services Web et des protocoles d'accès [5]. Il contient toutes les informations nécessaires à l'invocation d'un service Web : le nom du service, le nom de ses opérations (et leur paramètres) organisées sous forme de messages, le protocole de transport utilisé (souvent HTTP), l'encodage des données et l'adresse URL de localisation du service. Une description WSDL définit un service comme étant une collection de ports (i.e. endpoints). Elle se compose des parties suivantes (cf. Fig. 2.3) :

1. **types**, qui fournit la définition des types de données utilisés pour la description des messages échangés ;
2. **messages**, qui représente une définition abstraite des données échangées et leurs types associés ;
3. **portType**, qui identifie un ensemble d'opérations abstraites où chaque opération est composée d'un ou plusieurs messages ;
4. **binding**, qui spécifie le protocole concret et les spécifications de format de données pour les opérations et les messages définis par un portType particulier ;
5. **liaisons et ports de communication**, qui permet de lier les opérations (ou un portType) au protocole de transport sous-jacent.

SOAP

SOAP est un protocole de transport basé sur XML permettant de spécifier les échanges de messages [11]. SOAP se situe au dessus des autres protocoles réseaux (e.g. HTTP, SMTP) qui encapsulent les messages SOAP dans leurs propres messages. Un message SOAP a deux parties : (1) une partie entête optionnelle utilisée pour transférer les données d'authentification, ou de gestion de sessions, (2) le corps du message qui a en charge l'encodage des noms des opérations, de leurs paramètres ainsi que du résultat retourné.

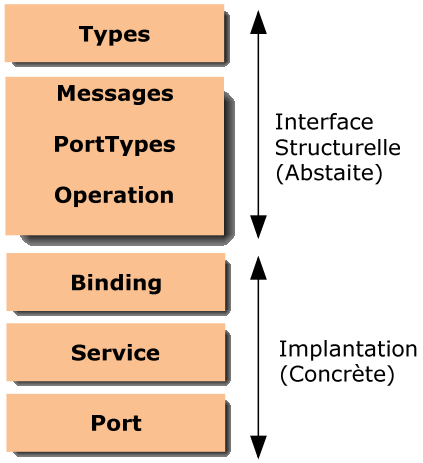

FIGURE 2.3 – Structure d'une description WSDL

UDDI

UDDI est un annuaire pouvant référencer des services Web [12]. Il contient un ensemble de fichiers de description de services Web, utilisant le langage WSDL. La communication entre un client et un service passe par une phase de découverte et de localisation de ce service, à l'aide du protocole SOAP et des annuaires UDDI.

2.2.3 Composition de services Web

La composition de services Web est la plus importante fonctionnalité assurée par une architecture SOA. Celle-ci offre un environnement homogène pour la composition dans la mesure où toutes les parties de la composition sont des services idéalement décrits de la même façon et communiquant par les mêmes standards d'échange de messages. La composition permet de combiner des services pour former un nouveau service dit composé ou composite. L'exécution d'un service composé implique des interactions avec des services partenaires en faisant appel à leurs fonctionnalités. Le but de la composition est avant tout la réutilisation de services (simples ou composés) et de préférence sans aucune modification de ces derniers.

Nous pouvons distinguer deux méthodes de composition de services Web :

Orchestration de services Web L'orchestration décrit la manière dont les services Web peuvent interagir ensemble au niveau des messages, incluant l'ordre d'exécution des messages et la logique métier. Dans l'orchestration, un seul processus (appelé orchestrateur) est responsable de la composition et contrôle les interactions entre les différents services. Cet orchestrateur coordonne de manière centralisée les différentes opérations des services partenaires qui n'ont aucune connaissance de cette composition. BPEL [1] est reconnu comme un langage standard d'orchestration de services Web.

Chorégraphie de services Web La chorégraphie décrit une collaboration entre services pour accomplir un certain objectif. A l'inverse de l'orchestration, la chorégraphie ne repose pas sur un seul processus principal. C'est une coordination décentralisée où chaque participant est responsable d'une partie du workflow, et connaît sa propre part dans le flux de messages échangés. WS-CDL [13] est l'un des langages de description de chorégraphie pour les services Web. Il peut être considéré comme un complément à un langage d'orchestration tel que BPEL en lui apportant un modèle global nécessaire pour assurer la cohérence des interactions des services partenaires, et pour prendre en considération les éventuelles situations de compétition entre partenaires.

En résumé, la différence majeure entre l'orchestration et la chorégraphie est que l'orchestration offre une vision centralisée de la composition, alors que la chorégraphie offre une vision globale et plus collaborative de la coordination.

2.3 Le langage BPEL

BPEL, tout d'abord nommé BPEL4WS [14] et WS-BPEL [1] depuis 2005, s'est imposé comme le langage standard OASIS [2] d'orchestration de services Web. Il est largement utilisé dans le cadre de la mise en œuvre d'une architecture orientée services. Le langage BPEL permet de décrire à la fois :

i. **l'interface comportementale**, via la définition d'un processus abstrait spécifiant les échanges de messages entre différents partenaires ;

ii. **l'orchestration**, via la définition d'un processus privé exécutable [3] qui représente la nature et l'ordre des échanges entre partenaires.

BPEL se caractérise par rapport aux autres langages (d'orchestration) par :

- sa gestion des exceptions, en particulier, des fautes et des événements ;
- l'exécution parallèle des activités et la synchronisation des flots ;
- la description des transactions [4] contextuelles (stateful) et de longue durée ;
- son mécanisme de compensation qui est très utile pour les transactions de longue durée ;
- sa gestion de la corrélation des messages.

Notons qu'un processus BPEL est directement exécutable par un moteur d'orchestration de BPEL (e.g. activeBPEL [15], Oracle BPEL Process Manager [16]).

BPEL s'appuie sur le langage WSDL puisque chaque processus BPEL est exposé comme un service Web via une interface WSDL. BPEL utilise les opérations, les données ainsi que les liens des partenaires décrits dans son interface WSDL. Ce dernier décrit aussi tous les éléments nécessaires à un processus BPEL pour interagir avec ses partenaires, à savoir, l'adresse des partenaires, les protocoles de communication et les opérations disponibles.

2. Organization for the Advancement of Structured Information Standards, `http://www.oasis-open.org`
3. Nous utilisons le terme processus BPEL pour désigner un processus BPEL exécutable.
4. La transaction est une suite d'actions dont l'exécution est cohérente, atomique, isolée des autres transactions et dont les effets sont durables.

En outre, BPEL fait appel aux standards WSDL, SOAP, UDDI, mais aussi aux autres standards de services Web (cf. Fig. 2.4) tels que :

- **WS-Addressing**, qui est un mécanisme permettant de transporter des messages SOAP de façon bidirectionnelle, en mode synchrone ou asynchrone [17] ;
- **WS-Policy**, qui est une extension de WSDL permettant d'exprimer les fonctionnalités et les caractéristiques générales des partenaires (e.g. politiques d'usage, gestion des transactions, fiabilité, sécurité) [18] ;
- **WS-Security**, qui est une extension de SOAP permettant de sécuriser les échanges de messages [19] ;
- **WS-Reliable Messaging**, qui est un protocole permettant aux messages d'être délivrés de manière fiable entre différentes applications réparties, dans le cas de pannes de composants, de réseaux ou de systèmes [20] ;
- **WS-Transactions**, qui définit des mécanismes interopérables permettant de réaliser des transactions entre différents domaines de services Web [21].

La description d'un processus BPEL contient quatre parties principales : (i) la déclaration des variables utilisant des types décrits ou importés dans l'interface WSDL, (ii) la description des partenaires (iii) la déclaration des gestionnaires de fautes qui sont déclenchés après une exception, (iv) l'activité principale qui décrit le comportement du processus.

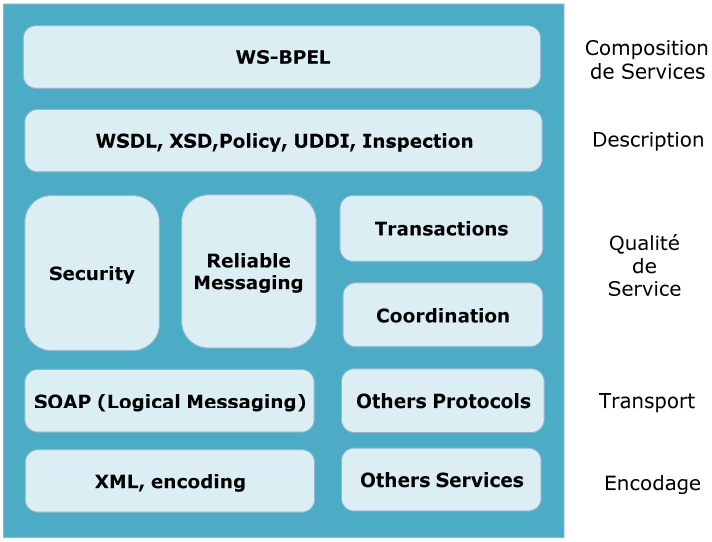

FIGURE 2.4 – BPEL dans l'architecture des services Web

2.3.1 Les activités de BPEL

Les activités, pouvant être effectuées par un processus BPEL, sont classées en trois catégories : activités basiques (ou de base), activités de communication et activités structurées [1].

Les activités basiques

Les activités de base sont des activités simples. Ce sont les activités : `<empty>`, `<throw>`, `<exit>`, `<assign>` et `<wait>`.

`<empty>` modélise une absence d'activité ou une activité vide.

`<throw>` permet de signaler une faute interne au processus BPEL.

`<exit>` termine instantanément l'instance d'un processus BPEL.

`<assign>` est utilisée pour mettre à jour les variables du processus BPEL en leur affectant de nouvelles données.

`<wait>` permet de définir une durée d'attente ou une échéance. Un seul critère d'expiration doit être spécifié à la fois.

Les activités de communication

Les activités de communication sont : `<receive>`, `<reply>`, et `<invoke>`. Elles sont utilisées pour l'échange des messages entre un processus BPEL, son client et ses partenaires.

`<receive>` permet d'attendre une invocation d'un service partenaire. Elle peut créer une instance d'un processus BPEL et le préparer ainsi à l'arrivée d'une réponse.

`<reply>` envoie un message en réponse à l'invocation d'un partenaire.

`<invoke>` permet d'invoquer une opération d'un service partenaire. Elle peut servir à la fois pour une communication synchrone ou asynchrone. L'activité <invoke> asynchrone joue le même rôle qu'une activité `<reply>` ; tandis que l'activité <invoke> synchrone invoque une opération d'un service partenaire mais attend sa réponse.

Les activités structurées

Les activités structurées décrivent l'ordre d'exécution de leurs sous-activités. Les activités `<sequence>`, `<if>`, `<while>` et `<repeatUntil>` permettent un contrôle séquentiel. L'activité `<flow>` permet quant à elle une exécution parallèle de ses activités ainsi que leur synchronisation. Enfin, l'activité `<pick>` définit un choix contrôlé par l'arrivée d'un événement.

`<sequence>` permet de définir une collection d'activités pouvant être exécutées séquentiellement.

`<while>` permet une exécution répétée de sa sous-activité tant que sa condition booléenne est satisfaite au début de chaque itération.

`<repeatUntil>` permet aussi une exécution répétée de sa sous-activité mais jusqu'à la satisfaction de sa condition.

`<if>` définit une liste ordonnée de choix conditionnels permettant de sélectionner une seule activité à exécuter.

`<pick>` attend qu'un message particulier (un événement extérieur) arrive ou qu'une alarme soit déclenchée. Quand l'un des deux événements se produit, l'activité associée est ainsi exécutée.

`<flow>` permet une exécution parallèle d'un ensemble d'activités et de spécifier un ordre partiel d'exécution. Une ou plusieurs de ses sous-activités seront donc concurrentes. Les liens définis dans cette activité permettent de synchroniser et/ou de spécifier des dépendances temporelles entre les sous-activités de `<flow>`.

Notons qu'un exemple d'un processus BPEL (service de prêt `loanApproval`) est fourni en Annexe A.

2.3.2 Gestion des données

Un des principaux défis dans l'intégration des services Web, et en particulier des processus métier, est de traiter les interactions contextuelles (stateful). Ainsi, il est nécessaire pour tout langage d'orchestration de fournir les moyens nécessaires pour traiter l'état d'une instance d'un processus métier.

Les processus BPEL permettent de spécifier ces interactions contextuelles impliquant l'échange de messages entre partenaires. L'état d'une instance d'un processus métier inclut les messages échangés ainsi que les données intermédiaires utilisées dans la logique métier et dans les messages envoyés aux partenaires. Le maintien de l'état de cette instance de processus nécessite l'utilisation de variables. En plus, les données des états ont besoin d'être extraites et utilisées de façon adéquate pour contrôler le comportement de l'instance du processus nécessitant des expressions sur les données. Enfin, la mise à jour de l'état du processus métier nécessite la notion d'assignation (de nouvelles valeurs aux variables en utilisant des affectations). BPEL fournit ces fonctions pour les types de données XML et les types de messages WSDL.

2.3.3 Corrélation des messages

La corrélation des messages est le mécanisme de BPEL permettant à des processus de participer aux conversations. Quand un message arrive à un processus BPEL, il doit être délivré à une nouvelle instance ou à une instance déjà existante. Le rôle de la corrélation des messages est donc de déterminer à quelle conversation un message appartient, et d'instancier/localiser une instance de processus. BPEL définit un ensemble de corrélation (i.e. `<correlationset>`) comme un ensemble de propriétés partagées par tous les messages échangés dans un groupe corrélé. Cet ensemble est employé pour lier les messages d'entrée et de sortie à une instance de processus BPEL grâce aux valeurs de corrélation contenues dans ces messages. Il peut être déclaré dans l'élément `<process>` de BPEL ou dans une activité `<scope>`. Chaque ensemble de corrélation est identifié par un nom qui sera utilisé dans une activité de communication.

2.3.4 Les scopes

BPEL permet de structurer un processus métier de manière hiérarchique sous forme de scopes imbriqués. L'activité `<scope>` fournit un contexte influant sur l'exécution de ses activités. Ce contexte inclut des variables, des liens partenaires et des ensembles de corrélation. Il a ses propres gestionnaires

de fautes, d'événements, de terminaison et de compensation. Ce contexte limite la visibilité de ces définitions aux activités englobées. Enfin, chaque activité `<scope>` a une activité principale qui décrit son comportement.

2.3.5 Compensation et gestion des erreurs

Les processus métier impliquent des transactions de longue durée qui sont basées sur des communications (i.e. échanges de messages) asynchrones. Ces transactions conduisent à un certain nombre de mise à jour pour les partenaires. Par conséquent, si une erreur se produit, il est nécessaire d'inverser les effets de certaines ou même de la totalité des activités antérieures. Cela est le rôle de la compensation.

Durant l'exécution d'un processus BPEL, des conditions d'exception peuvent se produire et doivent être gérées. Un service Web invoqué peut retourner une faute WSDL qui doit être traitée ; un processus BPEL lui-même peut avoir besoin de signaler une faute à des clients. Ajoutant à cela, que certaines erreurs peuvent aussi se produire au niveau de l'environnement d'exécution (des processus BPEL). Pour BPEL, la gestion des fautes peut être effectuée au niveau d'un processus et/ou d'un scope. Une activité `<scope>` définit une unité logique de travail recouvrable et réversible, pour lequel un ensemble de gestionnaires de fautes et de compensation peut être défini. Cela en vue de traiter les fautes interceptées, et de défaire le travail effectué dans un scope en cas d'erreurs. Notons que BPEL permet aussi d'associer des gestionnaires de fautes à une activité `<invoke>` (synchrone) pour traiter les fautes d'invocation.

2.4 Le test des logiciels

Les activités de validation et de vérification sont des activités qui visent à assurer qu'un système informatique développé satisfait bien les besoins exprimés et/ou les exigences de qualité requises. Elles sont complémentaires et très imbriquées aux différentes activités de développement : analyse, conception, programmation [4].

La vérification consiste à établir l'exactitude d'une spécification et/ou d'une implantation d'un système. Elle répond à la question : développe-t'on bien le système ? Elle inclut des inspection de spécifications et de programmes ainsi que de la preuve et du test. La validation de logiciel a pour but de répondre à la question : développe-t'on le bon système ? Elle se réfère aux besoins fonctionnels de départ et aux exigences globales de qualité (e.g. sûreté, vivacité). La validation reflète le point de vue de l'utilisateur. Elle consiste essentiellement en des revues et inspections de spécifications ou de manuels, des simulations et du prototypage rapide.

Le test est une activité de vérification. C'est une tâche importante dans le processus de développement d'un logiciel ou d'un système. L'objectif majeur du test est de concevoir des tests permettant, par exemple, de découvrir systématiquement différentes classes d'erreurs avec un minimum de temps et d'efforts, et de démontrer que certaines fonctionnalités sont bien atteintes par rapport aux attentes. La sélection des tests peut être basée sur le domaine d'entrée, sur les programmes, sur des spécifications, ou des modèles de fautes. Pour plus de précision, on se réfère à [4].

Dans la littérature, de nombreuses définitions du test existent. Nous nous contentons d'en donner deux définitions : Selon l'IEEE[5] : « *Le test est l'exécution ou l'évaluation d'un système ou d'un composant par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus.* ».

Selon [22] : « *Le test d'un logiciel est une activité qui fait partie du processus de développement. Il est mené selon les règles de l'assurance de la qualité et débute une fois que l'activité de programmation est terminée. Il s'intéresse aussi bien au code source qu'au comportement du logiciel. Son objectif consiste à minimiser les chances d'apparition d'anomalie avec des moyens automatiques ou manuels qui vise à détecter aussi bien les diverses anomalies possibles que les éventuels défauts qui les provoqueraient.* ».

Diverses classifications de techniques de tests ont émergé. Dans ce travail, nous classons le test selon trois axes [23] : (i) niveau de détail, (ii) niveau d'accessibilité (ou de connaissance de la structure de l'objet à tester), (iii) type de caractéristique ou de propriété à tester. Cette classification est illustrée dans la figure 2.5.

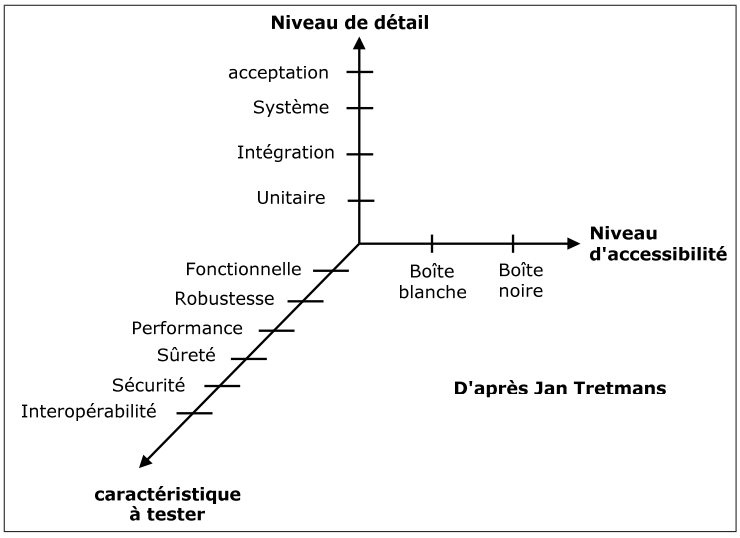

FIGURE 2.5 – Classification des tests

Classification selon le niveau de détail Cet axe identifie quatre niveaux de test :

- **test unitaire**, qui vérifie le fonctionnement de chaque composant en isolation du reste du système ;
- **test d'intégration**, qui vérifie les interfaces et les interactions entre composants ou systèmes intégrés ;
- **test système**, qui vérifie le comportement global du système, une fois intégré ;
- **test d'acceptation**, qui est conduit pour déterminer si un système satisfait ou non aux critères (besoins et exigences) d'acceptation.

5. Standard Glossary of Software Engineering Terminology, `http://www.ieee.org/portal/site`

Le **test de non régression** vient en complément à ces différents niveaux de test. C'est le test d'un système préalablement testé, après une modification, pour s'assurer que des défauts n'ont pas été introduits ou découverts dans des parties non modifiées du système.

Le test de régression est effectué quand le système ou son environnement est modifié. Il peut compléter, selon les modifications appliquées, le test unitaire, le test d'intégration et le test système en amont du test d'acceptation.

Classification selon le niveau d'accessibilité On peut distinguer deux types de techniques de conception de test (boîte blanche, boîte noire) dont sont issus les tests suivants :

- **test boîte blanche**, qui est un test fondé sur l'analyse de la structure interne du composant ou du système. Ce test est appelé aussi test de structure ou **test structurel** ;
- **test boîte noire**, qui est un test, fonctionnel ou non fonctionnel, qui se base sur aucune analyse (ou référence) de la structure interne du composant ou du système.

Nous ajoutons à ces deux techniques, la technique de conception de **test boîte grise** qui consiste à tester un système avec une connaissance limitée de sa structure interne.

Classification selon le type de caractéristique à tester Nous citerons ci-dessous quelques exemples :

- **test fonctionnel**, qui est basé sur l'analyse des spécifications d'une fonctionnalité d'un composant ou d'un système. Il sert à vérifier que les fonctions sont bien atteintes, par rapport aux attentes ;
- **test de performance**, qui permet de déterminer les performances d'un système, ou de vérifier si les performances annoncées dans la spécification du système sont bien atteintes ;
- **test de sûreté**, qui est effectué pour déterminer la sûreté d'un système (i.e. capacité d'un produit logiciel à obtenir des niveaux de risques acceptables [ISO 9126]) ;
- **test de robustesse**, qui s'intéresse au degré de résistance de l'implantation du système à des événements externes ou à des erreurs non prévues par la spécification de ce système ;
- **test d'interopérabilité**, qui évalue le degré d'interopérabilité de plusieurs implantations. Il implique le test des capacités et du comportement de l'implantation dans un environnement interconnecté.

Le **test de conformité** est un test fonctionnel qui consiste à vérifier qu'une implantation d'un système est conforme à sa spécification. Il permet donc de déterminer la conformité d'un composant ou d'un système à ses exigences. Le test de conformité est une approche boîte noire si un testeur externe ne peut observer que les sorties générées par l'implantation suite à la réception de données d'entrées. Le concept de conformité d'une implantation par rapport à une spécification est exprimé à l'aide d'une relation de conformité.

On distingue les notions de *faute*, de *défaillance* et d'*erreur* [4]. Lors de l'exécution d'un logiciel, une défaillance désigne un comportement observable mais qui n'est pas conforme à celui attendu. Une faute est une caractéristique du logiciel qui risque de provoquer une défaillance ou une erreur lors d'une exécution. Une erreur est la manifestation d'une faute dans l'état interne du logiciel, qui est

susceptible de causer une défaillance sauf si elle est traitée par le logiciel. Les relations de causalité entre ces trois notions sont comme suit : une faute peut provoquer une erreur qui à son tour peut provoquer une défaillance.

Avant de pouvoir tester un service composé, ce dernier doit d'abord être spécifié. Pour cela, nous passons en revue dans la prochaine section une partie des travaux qui ont abordé la modélisation et la spécification formelle de l'orchestration de services Web décrite en BPEL.

2.5 Modélisation de l'orchestration de services Web

Les méthodes formelles sont bien adaptées pour la spécification, la vérification et la validation de la composition de services Web [24, 25]. Dans ce contexte, il existe de nombreux langages de spécification et de modèles formels permettant de décrire l'orchestration de services. Depuis quelques années, une variété de travaux a été publiée, visant à décrire formellement la composition de service Web, et à proposer des méthodes formelles pour la vérification et la validation des services composés [26, 27, 28]. Maurice et al. [27] présentent différentes approches de composition de services Web, alors que van Breugel et al. [28] donnent un aperçu des différents modèles qui ont été proposés pour BPEL. Nous pouvons classer ces modèles selon les familles suivantes : réseaux de Pétri, automates, algèbre de processus et machine d'états abstraite. Dans la suite, nous aborderons, pour chaque famille, les travaux récents et majeurs qui y sont associés. Nous discuterons ensuite ces différents modèles en les comparant à notre modèle WS-TEFSM.

2.5.1 Réseau de Pétri

Un réseau de Pétri est un modèle mathématique servant à modéliser le comportement des systèmes qui ont la caractéristique d'être concurrents, asynchrones, distribués, parallèles, non-déterministes et/ou stochastiques. [29]. Il est représenté par un graphe biparti orienté reliant des places (représentant l'aspect local des états globaux) et des transitions (représentant des actions). Les réseaux de Pétri ont été largement utilisés pour modéliser les processus BPEL [28, 30]. Ainsi, chaque processus BPEL est transformé en un réseau de Pétri dont la sémantique a été définie formellement. Les outils et les techniques développés pour les réseaux de Pétri peuvent être utilisés et exploités dans le contexte de la composition de services Web. Une algèbre de services Web a été proposée dans [31] afin de définir des opérateurs de composition de services Web. Hamadi et al. ont utilisé les réseaux de Pétri comme sémantique formelle de cette algèbre.

De son côté, Christian Stahl [32, 33] a défini une autre sémantique de BPEL basée sur le modèle des Réseaux de Pétri. Cette sémantique est assez complète étant donné qu'elle décrit tous les comportements d'un processus BPEL, y compris la gestion des exceptions (e.g. fautes, événements et compensation). L'outil BPEL2PNML [32] a été développé pour transformer BPEL en réseaux de Pétri qui ont été fournis en entrée à l'outil de vérification WofBPEL [34]. Dans [33], l'outil BPEL2PN est utilisé pour transformer BPEL en réseaux de Pétri ainsi que le Model-Checker LoLA [35] pour la vérification des propriétés des processus BPEL décrites en logique temporelle CTL. Lohmann et al. [36] ont étendu les travaux [33] en utilisant les réseaux de Pétri pour décrire les interactions entre processus BPEL.

Hinz et al. [37] ont présenté la description formelle d'une sémantique en réseaux de Pétri pour BPEL. Ils ont développé un outil BPEL2PN pour transformer automatiquement un processus BPEL en réseaux de Petri. En conséquence, une variété d'outils de vérification de réseaux de Pétri ont pu être ainsi appliqués pour l'analyse automatique de ces processus. Quant à Dong et al. [38], ils ont proposé, pour le test de BPEL, de transformer un processus BPEL en un réseau de Pétri de haut niveau (HPN). Enfin, d'autres travaux (e.g. [39, 40, 41, 42]) ont utilisé le modèle des réseaux de Pétri colorés (CPN) [43] pour la modélisation de BPEL. Pour exemple, Kang et al. [39] ont proposé un modèle basé sur les CPNs pour la composition de services Web. Ils ont décrit une partie de la sémantique de BPEL en CPNs et ont ensuite utilisé l'outil CPNTOOLS [44] pour analyser et vérifier la composition de services.

2.5.2 Automates

Zheng et al. [45, 46, 47] ont défini une sémantique opérationnelle de BPEL pour le test de la composition de services Web. Cette sémantique a été décrite par un automate, appelé WSA (Web Service Automata), qui est une extension des machines de MEALY. Ce modèle WSA a servi comme modèle intermédiaire et a été codé en XML pour être transformé ensuite en langage d'entrée des Model-Checkers NuSMV [48] et SPIN [49]. WSA est assez adapté une grande partie de ses constructions, à l'exception des activités temporelles (e.g. `<wait>`, `<onAlarm>`) [1] étant donné que c'est un modèle non temporisé. Par contre, les données complexes, les valeurs concrètes et les prédicats de BPEL ont été abstraites afin de faciliter le Model-Checking.

Le modèle WSTTS (Web Service Timed State Transition Systems) a été proposé par Kazhamiakin et al. [50, 51] pour décrire les comportements temporisés d'un processus BPEL. Intuitivement, WSTTS est une machine d'états finis enrichie avec un ensemble de variables d'horloge mais ne considérant pas les variables discrètes ; En conséquence, ce modèle ne permet pas donc de décrire les variables d'un processus BPEL, ni la gestion de la corrélation des messages.

Dans [52], une transformation de BPEL en automate (aDFA pour annotated Deterministic Finite State Automata) a été proposée par Mahleko et al.. Cette modélisation de BPEL en aDFA a été utilisée par Wombacher et al. [53] pour la découverte de services. Cependant, ce modèle aDFA ne permet pas de décrire ni les aspects temporels des processus BPEL, ni ses variables ou ses gestionnaires d'exceptions.

Un autre modèle d'automate, appelé STS (State Transition System), a été défini et utilisé par Pistore et al. [54] comme sémantique formelle de BPEL. La transformation de BPEL en STS a été implantée dans l'outil BPEL2STS. Notons que ce modèle ne prend pas en compte ni les données, ni le temps et ne décrit qu'une partie de BPEL. Enfin, nous citons les travaux de Yan et al. [55] qui ont présenté une méthode automatique de modélisation des comportements d'un service composé par le biais de systèmes à événements discrets DES (Discrete Event Systems). Yan et al. ont transformé une description BPEL en un système DES et ont utilisé ce modèle pour le diagnostic et la supervision (ou le monitoring) des processus BPEL.

Dans [56, 57], Bultan et al. ont introduit une plateforme pour l'analyse de la composition de processus BPEL communiquant via des messages XML asynchrones, et la vérification de leurs propriétés. Cette plateforme permet, en premier lieu, de transformer chaque processus BPEL en un type particulier d'automates gardés appelés GFSA (Guarded Finite State Automata) dont chaque transition peut être associée à une garde sous la forme d'une expression XPath [58].Ces automates ont été ensuite transformés en une spécification Promela [59]. L'outil SPIN [6] a servi ainsi pour vérifier si la composition des processus BPEL satisfait certaines propriétés décrites en logique temporelle linéaire (LTL). Ces mêmes auteurs ont décrit, dans [57], leur outil WSAT permettant de transformer BPEL en GFSA — utilisée comme une représentation intermédiaire — et ensuite GFSA en Promela.

Shin NAKAJIMA [60] a proposé d'utiliser les techniques de Model-Checking pour l'analyse des services composés décrits en BPEL. Ses auteurs ont décrit le comportement d'un processus BPEL par une variante d'automate EFA (Extended Finite State Automaton). Cet automate est transformé en une spécification Promela qui est automatiquement analysée par le Model-Checker SPIN. Notons que ce modèle formel ne permet pas de décrire toutes les constructions de BPEL, en particulier, les activités temporelles, la gestion des exceptions (fautes, événements, compensation) et la terminaison. Enfin, García-Fanjul et al. [61, 62] ont transformé un processus BPEL en une spécification Promela qui a été utilisée comme une entrée du Model-Checker SPIN afin de dériver une suite de tests pour le test des processus BPEL.

2.5.3 Algèbre de processus

Les algèbres de processus sont une famille de langages formels permettant de modéliser des systèmes concurrents ou distribués. Différentes algèbres de processus ont été définies telles que LOTOS [63], π-calcul, le calcul des systèmes communicants (CCS). Ces algèbres de processus sont généralement représentées par un système de transitions étiquetées (LTS) où la relation de transition est définie par une collection de règles et d'axiomes. Rampacek et al. [64] ont proposé une méthodologie outillée permettant la modélisation formelle et l'analyse des processus BPEL. Rampacek et al. ont défini une formalisation de la sémantique de BPEL par une algèbre de processus temporisés appelée (Atp). Les comportements d'un processus BPEL sont décrits par des systèmes de transition à temps discret qui sont obtenus par une simulation exhaustive de cette algèbre à l'aide de l'outil WSMod. Ces modèles ont été ensuite analysés par la technique du Model-Checking en utilisant la boîte à outils CADP [65], en particulier, le Model-Checker Evaluator.

Foster et al. [66] ont présenté une approche de spécification, de modélisation et de validation des comportements des services composés. La sémantique de BPEL a été décrite par une notation d'algèbres de processus appelée FSP (Finite State Processes) représentant un système de transitions étiquetées (LTS). Un processus BPEL est ainsi transformé en une expression FSP qui est fournie en entrée à l'outil LTSA (Labelled Transition System Analyser) pour la génération d'un modèle LTS et son analyse. Ce modèle LTS est utilisé ensuite dans la validation et la vérification formelle de la composition. Notons que cette formalisation ne considère que le flot de contrôle des services composés sans prise en compte de leur aspect temporel. En plus, elle ne décrit pas toutes les constructions de BPEL, en particulier, la gestion de la corrélation des messages et des exceptions.

6. SPIN est un outil de vérification formelle des systèmes distribués décrits en langage Promela permettant de modéliser des systèmes distribués asynchrones par le biais d'automates à files d'attente. Les propriétés à vérifier sont décrites par des formules en logique linéaire temporelle LTL.

Dans [67, 68], a été introduit une algèbre de processus, appelée BPEL-calculus, pour modéliser une grande partie des activités de BPEL. La syntaxe de BPEL-calculus a été décrite par une grammaire, et respectivement sa sémantique par une collection d'axiomes et de règles. Cette algèbre a été utilisée dans la vérification des processus BPEL. De leur coté, Weidlich et al. [69] ont proposé une formalisation d'une partie de BPEL par une algèbre π-calcul. Enfin, Salaun et al. [70] ont défini deux méthodes de transformation de BPEL en langage LOTOS [63]. La plupart des activités de BPEL, y compris la gestion des exceptions (fautes et événements) et la compensation, ont été prises en compte. Après avoir transformé BPEL en LOTOS, la plateforme CADP a été exploitée pour la vérification des processus BPEL.

2.5.4 Machine d'états abstraite

Les machines d'états abstraites ASMs (Abstract State Machine) ont été utilisées pour modéliser une grande variété de langages. Une machine ASM basique est un ensemble de règles de transition où chaque transition est constituée d'une expression booléenne et d'un ensemble fini d'affectations. Ces règles déterminent quelle transition la machine ASM peut s'exécuter en passant d'un état source à un autre état. Ce dernier est obtenu en effectuant toutes les affectations des règles de transitions dont les expressions booléennes sont évaluées à $true$ [71].

Farahbod et al. [72, 73] ont décrit une sémantique abstraite assez complète pour BPEL en termes de machines d'états abstraites (ASMs) distribuées et temps réel. Les différents aspects de BPEL ont été pris en compte, tels que la corrélation des messages, la compensation des activités et la gestion des fautes et des événements. Notons que pour traiter les aspects temporels de BPEL, une notion abstraite du temps global a été introduite au modèle ASM en plus des contraintes supplémentaires sur les séquences de transitions qui ont été ajoutées. Ce modèle a servi dans la vérification formelle de la composition de services. Il a aussi été utilisé dans la validation expérimentale en rendant cette sémantique exécutable par l'usage du langage AsmL (Abstract State Machine Language) [74]. Ce modèle ASM a été étendu dans [75, 76, 77] afin de couvrir tous les aspects de BPEL ; en particulier Dirk Fahland [75] qui a fourni une sémantique complète de BPEL.

2.5.5 Autres formalismes

Butler et al. [78] ont défini un langage stAC (Structured Activity Compensation) afin de l'utiliser pour spécifier l'orchestration des activités dans le cadre des transactions de longue durée. Ce type de transactions utilise la compensation pour la gestion des exceptions. stAC considère les comportements séquentiels ou parallèles ainsi que la compensation et la gestion des exceptions. La notation B a été combinée au langage stAC pour spécifier les données des transactions. stAC a été utilisé pour décrire la sémantique d'une partie de BPEL. De leur coté, Yuan et al. [79] ont défini un nouveau modèle appelé BFG (BPEL Flow Graph), qui est une extension du graphe de flot de contrôle (CFG), pour représenter les processus BPEL sous forme d'un modèle graphique. Ce modèle BFG a servi dans la génération de cas de test pour BPEL. Ces cas de test sont construits en combinant les chemins de test générés en traversant le modèle BFG, et les données générées par résolution de contraintes. Ce modèle BFG permet de spécifier le flot de contrôle de BPEL, le flot de données mais qu'une partie de la sémantique de BPEL.

Il existe dans la littérature d'autres travaux de modélisation de la composition de services par des diagrammes UML [80]. Nous donnons, pour exemple, les travaux de Cambronero et al. [81] qui ont utilisé les diagrammes UML pour décrire les comportements de processus BPEL ainsi que les contraintes temporelles qui y sont associées. Ils ont aussi proposé une méthode automatique de transformation de ces diagrammes UML en BPEL.

2.5.6 Discussion

Comme nous l'avons cité ci-dessus, Christian Stahl [32, 33] a défini une sémantique assez complète de BPEL basée sur le modèle des Réseaux de Pétri, ainsi que Farahbod et al. [72, 73] qui ont décrit une sémantique complète de BPEL en termes de machines d'états abstraites (ASMs). Cependant, nous ne prenons pas en considération ces différents modèles — en particulier les réseaux de Pétri, les algèbres de processus et les machines d'états abstraites — car nous voulons tirer profit de notre expérience des méthodes formelles et des techniques de génération dans le domaine du test (qui sont associés aux modèles de machine d'états et/ou automates). Nous avons ainsi essayé d'adapter ces méthodes au domaine spécifique des services Web et de leur composition et/ou de définir de nouvelles méthodes.

De plus, de nombreux modèles proposés pour la modélisation de l'orchestration des services Web ne sont pas temporisés et/ou ne peuvent décrire qu'une partie de la sémantique de BPEL. En outre, ils n'ont pas été utilisé pour le test mais pour l'analyse, la vérification par Model-Checking, la découverte ou la supervision des services composés ; c'est le cas, en particulier, des modèles de réseaux de Pétri [32, 33, 37], WSTTS [50], aDFA [52], STS [54], DES [55], GFSA [56], EFA [60], Atp [64], FSP [66] et ASM [72]. D'autres modèles (e.g. WSA [45]) ont servi comme modèle intermédiaire au test de la composition de services mais par Model-Checking. Ajoutant à cela que l'utilisation de certains de ces modèles (e.g. algèbres de processus, réseaux de Pétri) pourrait présenter une difficulté de définition d'algorithmes applicables dans un outil de génération de test.

La définition de notre modèle temporisé, appelé WS-TEFSM, et son utilisation pour le test de l'orchestration de services Web sont motivées, d'une part, par le fait que BPEL (le langage standard d'orchestration) possède quelques éléments et activités temporels qu'il faut formaliser pour tester les propriétés temporelles d'un service composé, et d'autre part, par la nécessité de la gestion de terminaison et des exceptions dans BPEL. Le modèle WS-TEFSM, proposé et présenté dans le Chapitre 3, est assez expressif et bien adapté particulièrement pour décrire : (i) les aspects temporels de BPEL étant donné que c'est un modèle temporisé, (ii) la gestion de la terminaison et des exceptions grâce à son concept de priorité des transitions permettant de gérer la terminaison (forcée) des activités de BPEL.

Après avoir présenté les différentes travaux de modélisation de l'orchestration des services Web, nous allons présenter dans la section suivante, les différentes méthodes de test des services Web composés décrits en BPEL (i.e. les processus BPEL).

2.6 Test des services Web composés décrits en BPEL

En raison des caractéristiques complexes de la composition des services Web décrite en BPEL, en particulier, la corrélation des messages, la compensation des activités, et la synchronisation et la gestion des liens, les processus BPEL sont sujets à différents types d'erreurs. De plus, les processus BPEL peuvent utiliser des ressources supplémentaires à travers des appels de services partenaires. Pour toutes ces raisons et vu l'utilisation croissante des services Web, il est nécessaire de s'assurer du comportement correct de ces processus. Le test des processus BPEL est un moyen efficace pour détecter les comportements incorrects.

Au cours de ces dernières années, la communauté du test de logiciels a commencé à s'impliquer dans le domaine des services Web. Différents outils et techniques ont été développés pour le test des services Web, y compris la composition de services. Diverses approches de test de la composition de services ont été analysées par [82]. En raison de l'importance du rôle de BPEL dans la composition des services, et plus généralement, dans l'architecture orientée services (SOA), le test de BPEL est devenu, ces dernières années, un axe de recherche important. Dans cette section, nous nous intéressons en particulier au test de l'orchestration des services Web décrite en BPEL (qui est l'un des objectifs de cette thèse) et non au test de la chorégraphie. Nous allons aborder différents techniques de test de BPEL : (i) le test de l'interface WSDL, (ii) le test structurel ou boîte blanche, (iii) le test boîte noire ou fonctionnel.

2.6.1 Test de l'interface WSDL

Pour les applications Web et e-business, le test d'interface homme-machine revêt une importance croissante. C'est le cas aussi pour les services Web et de leur interface WSDL. Le test d'un service Web (de base ou simple) peut être vu comme un test boîte noire de son implantation. Les cas de test sont ainsi dérivés à partir de son interface WSDL considérée comme étant une spécification de ce service. Nous pouvons aussi appliquer cette approche de test à l'orchestration de services Web en ne considérant que l'interface WSDL du service composé. Et cela sans aucune considération de ses interactions avec ses partenaires ; seul le client du processus BPEL est pris en compte.

Différents travaux académiques ont abordé le test de conformité, d'interopérabilité et de robustesse des services Web considérés comme des boîtes noires [83, 84, 85, 86, 87, 88, 89, 90, 91]. Salva et al. [83] ont proposé une méthode de test automatique des descriptions WSDL. Cette méthode est en mesure de générer des cas de test automatiquement, et de vérifier la gestion des exceptions, la gestion des sessions ainsi que l'existence des opérations. Salva et al. ont aussi décrit un Framework de test permettant d'exécuter les tests et de fournir un verdict de test. De leur coté, Bartolini et al. [84] ont proposé une approche de test automatique des descriptions WSDL, combinant à la fois la couverture des opérations (de l'interface WSDL) ainsi que la génération des données de test. En outre, ils ont défini une architecture de test intégrant deux outils existants : SoapUI [92], qui est un outil de test manuel des services Web, et TAXI [93, 94], qui permet la dérivation automatique des instances XML conformes et représentatives d'un schéma XML. Dans ce travail, la génération de cas de test est basée sur des critères de couverture ainsi que l'application de certaines heuristiques. Ces dernières visent, en particulier, à combiner de façon variée les différents éléments instanciés (à partir d'un schéma XML d'un document WSDL), et à varier les cardinalités ainsi que les valeurs de données de ces instances.

Dans [85], a été présenté un Framework de test des services Web utilisant le langage de spécification et d'implémentation de test TTCN-3 [95]. Pour les besoins du test, un ensemble de règles de transformation a été défini pour dériver une suite de tests abstraits en TTCN-3 à partir d'une description WSDL. Bai et al. [86] ont proposé une méthode de génération automatique de cas de test à partir d'une description WSDL décrivant l'interface d'un service Web (opérations offertes et données transmises). Tout d'abord, le document WSDL est analysé et transformé en une structure d'arbre DOM [96]. Ensuite, les cas de test sont dérivés à partir de la génération des données et des opérations WSDL.

Bertolino et al. ont abordé, dans [87], le test d'interopérabilité entre services Web. Ils ont proposé d'enrichir la description WSDL d'un service par un diagramme UML, appelé PSM (Protocol State Machine), modélisant les interactions possibles entre le service et son client. Les cas de test sont ainsi dérivés à partir du diagramme PSM et sont ensuite exécutés par le Framework, appelé Audition Framework, qui a été décrit dans [97].

Enfin, quelques outils de test des services Web ont émergé ces dernières années tels que soapUI [92], TestMaker [98] et WSUnit [99]. Ces outils facilitent la supervision et le débogage de l'exécution des services Web, la validation syntaxique et sémantique des descriptions WSDL ainsi que la vérification des messages SOAP.

L'interface WSDL ne fournit aucune description des comportements d'un service Web (composé) étant donné qu'elle ne décrit que les données ainsi que les différentes opérations offertes par ce service. Tester WSDL revient donc à tester indépendamment les opérations d'un service mais ne permet pas de tester efficacement l'orchestration de services.

2.6.2 Test structurel ou test boîte blanche de BPEL

Le test structurel ou le test boîte blanche, permet de valider la structure d'un programme ou d'un composant logiciel. Il se base sur un modèle du code source du programme ou une représentation interne de sa structure. Ce test fait appel aux techniques de couverture, et en particulier, aux critères de couverture sur le flot de contrôle et le flot de données. Pour le test structurel, on distingue deux approches : (i) approche statique reposant, entre autres, sur l'analyse du programme, (ii) approche dynamique reposant sur l'exécution du programme.

De nombreux travaux [100, 101, 102, 79, 38] se sont intéressés au test structurel de BPEL. Bartolini et al. [100] ont tout d'abord étudié la possibilité de recourir à la modélisation du flot de données pour le test structurel des services Web composés. Liu et al. [101] ont proposé une approche de test structurel des processus BPEL. Cette approche est fondée sur un modèle de graphe de flot de contrôle BCFG (pour WS-BPEL Control Flow Graph), qui prend en compte les caractéristiques de la concurrence et de synchronisation de BPEL, tout en adaptant la notation BPMN pour représenter le graphe de flot de contrôle du service composé. En se basant sur ce modèle BCFG, les chemins de test d'un processus BPEL peuvent être dérivés en traversant le modèle, comme dans [79], mais en évitant les boucles ou en les traversant au plus une seule fois. Les conditions de chemins sont ensuite collectées et analysées pour éliminer les chemins infaisables et ainsi déterminer les cas de test pour valider la composition.

En utilisant le modèle du graphe de flot de BPEL, noté BFG (cf. § 2.5.5), Yuan et al. [79] ont présenté un processus BPEL (ou plutôt une partie de la sémantique de BPEL) sous forme graphique, et ont proposé une méthode de génération de test. Les chemins de test concurrents de BPEL sont dérivés en traversant le graphe BFG alors que les données de chaque chemin sont engendrées en utilisant une méthode de résolution de contraintes. Cette méthode a été implantée dans l'outil BPELTester [103], plus précisément, dans l'algorithme d'exploration des chemins de test.

Une méthode similaire de génération de test pour BPEL, fondée sur l'analyse des chemins concurrents, a été proposée par Yan et al. dans [102]. Cette méthode utilise un graphe de flot de contrôle étendu (XCFG) pour représenter un processus BPEL. Ensuite, elle dérive à partir de ce graphe XCFG tous les chemins de test séquentiels qui sont combinés pour former tous les chemins concurrents. Un solveur de contraintes est alors utilisé pour résoudre les contraintes de ces chemins de test pour générer les tests faisables ou possibles.

Dong et al. [38] ont défini une méthode de test de BPEL utilisant le modèle des réseaux de Pétri de haut niveau HPN (cf. § 2.5.1). En analysant la structure d'un processus BPEL, ce dernier peut être transformé en réseaux HPN qui sont ensuite vérifiés par utilisation d'un simulateur HPN. Les méthodes de génération de test, les techniques de réduction et de couverture de test associées aux réseaux HPN sont alors employées pour la génération et l'optimisation des cas de test.

En plus des travaux cités ci-dessus, et dans le même contexte du test structurel de la composition de services Web, nous pouvons distinguer deux approches : (i) test unitaire de BPEL, (ii) test de BPEL par Model-Checking.

Test unitaire de BPEL

Quelques travaux (e.g. [104, 105]) ont abordé le test unitaire de BPEL et ont proposé un Framework de test. Mayer et al. [104] ainsi que Philip Mayer [106], ont proposé une approche pour la création d'un Framework de test unitaire en approche boîte blanche de BPEL. Ils ont décrit leur propre Framework de test BPELUnit [107] qui utilise son propre langage de test ainsi que les données de test (sous forme XML) afin de décrire les interactions du processus BPEL qui doivent être effectuées par un cas de test. Ce Framework permet la simulation des partenaires du processus BPEL sous test, l'invocation de ce processus par l'intermédiaire de messages SOAP ainsi que la collecte des résultats de test. Il prend en charge l'édition, l'organisation, la gestion, le déploiement ainsi que l'exécution automatique de tests. Ajoutant à cela, qu'il permet de tester les interactions synchrones ou asynchrones d'un processus BPEL avec un nombre variable de partenaires. BPELUnit peut être étendu pour supporter de nouveaux encodages de SOAP, moteurs BPEL et environnements d'exécution. Malheureusement, les tests et les données échangées ne sont éditées que manuellement, en plus de l'absence de toute méthode automatique de génération de tests.

L'approche définie dans [105] contient quelques idées intéressantes concernant ce type de test de BPEL. Li et al. [105] ont proposé un Framework générique de test unitaire de BPEL incluant à la fois un modèle abstrait d'un processus BPEL, une architecture de test, la gestion des tests et la conception d'un ou de plusieurs processus testeurs. Ils ont défini une architecture de test générique basée sur la simulation des services partenaires, ainsi que ses différentes variantes (architecture centralisée, architecture distribuée, architecture de test de plusieurs instances d'un processus BPEL). Ces mêmes

auteurs ont fourni un exemple d'implantation de leur Framework de test en utilisant BPEL comme langage de spécification et de description du test (chaque partenaire du service sous test, i.e. processus BPEL, est décrit par un processus BPEL testeur). Ce Framework a servi pour guider le test manuel et la génération de tests. Notons que dans cette approche, ni le déploiement du processus sous test, l'exécution des tests, la configuration des services testeurs ou encore les données du test (données échangées entre processus sous test et partenaires) n'ont été détaillés.

D'autres approches plus pratiques consistent à tester BPEL en simulant l'exécution du processus BPEL. Dans ce cas, les moteurs BPEL s'exécutent en mode simulation (pas de déploiement de processus BPEL) et non pas en mode réel. Dans ce type d'approche, tester BPEL revient à injecter ou à extraire directement des données au lieu de communiquer avec les processus BPEL sous test par le biais de messages SOAP. Ces approches sont utilisées pour le test manuel de BPEL et limitées à un seul moteur BPEL ; elles ne tiennent pas en compte la complexité de l'encodage SOAP ni la transmission des messages. Dans ce cadre, il existe une variété d'outils tels que Oracle BPEL Process Manager [16] ou Parasoft SOAtest [108], qui prennent en charge le test (boîte blanche pour la plupart) des processus BPEL. Pour exemple, Oracle BPEL Process Manager fournit un Framework de test automatisé pour la création et l'exécution des tests pour des processus BPEL [109].

Les Frameworks présentés ci-dessus, en particulier BPELUnit [107], facilitent le test unitaire en approche boîte blanche des processus BPEL. Pour certains, ils offrent des fonctionnalités d'édition des cas tests (y compris des données de test), d'encodage/décodage de messages SOAP, d'exécution de tests et d'émission du verdict de test. Cependant, aucun de ses outils ou Frameworks ne permet une génération automatique de tests.

Test de BPEL par Model-Checking

Le Model-Checking est une technique de vérification formelle permettant de déterminer si le modèle d'un système satisfait certaines propriétés (qui sont souvent formulées en logique temporelle). Cette technique a été utilisée dans le domaine du test, et en particulier, pour les critères de couverture structurelle. L'idée de base est d'utiliser un Model-Checker pour dériver un cas de test en décrivant les critères (ou les objectifs) de test comme une négation des propriétés à vérifier. Chaque cas de test est engendré à partir d'un contre exemple (i.e. une trace d'exécution du modèle qui ne satisfait pas les propriétés à vérifier). Certains travaux (e.g. [46, 47, 62, 61, 110]) ont proposé de tester la composition de services Web en utilisant des techniques de Model-Checking et des critères de couverture structurelle.

Zheng et al. [46, 47] ont proposé un Framework de génération de test pour les compositions BPEL. La sémantique de BPEL a été décrite par un modèle formel intermédiaire, i.e. l'automate WSA (cf. § 2.5.2). L'approche proposée transforme un processus BPEL en un automate WSA, qui est transformé à son tour soit en SMV (le langage d'entrée du Model-Checker NuSMV [48]), soit à Promela (le langage d'entrée de SPIN [49]). Ces deux Model-Checkers (NuSMV et SPIN) sont ainsi utilisés comme outil de génération de cas de test où les critères de couverture (e.g. état, transition, tous-les-chemins) sont décrits par une logique temporelle LTL ou CTL. Ajoutant à cela, les travaux de García-Fanjul et al. [61, 62] qui ont consisté à transformer directement les processus BPEL en Promela sans utilisation de modèle intermédiaire. SPIN [49] et le critère de couverture des transitions ont été utilisés pour la génération de test.

Pour le test par Model-Checking, les critères de couverture sont exprimés par des formules en logiques temporelles. La génération de tests se réduit alors à un problème de satisfaction de formules logiques. Les techniques de réduction utilisées par les Model-Checkers peuvent servir à éviter le problème d'explosion pendant la génération (à la volée) de tests. Par contre, la contrainte, dans ce cas, est d'utiliser les langages d'entrée de ces Model-Checkers (e.g. Promela pour SPIN [49]) pour décrire la sémantique de BPEL ou l'orchestration de services ; de tels langages ne sont capables de décrire qu'une partie de BPEL et de ses données complexes ; même si quelques travaux ont proposé de décrire cette sémantique de BPEL avec leur propre modèle formel, et de le transformer ensuite en langage d'entrée d'un Model-Checker.

2.6.3 Test fonctionnel ou test boîte noire de BPEL

Le test fonctionnel, qui fait partie des méthodes de test boîte noire, permet de vérifier l'adéquation du logiciel aux contraintes (ou exigences) définies dans les spécifications. Il n'examine pas la structure du programme mais plutôt son comportement fonctionnel. Il évalue la réaction du logiciel par rapport à certaines données d'entrées. Dans le contexte de l'orchestration des services Web, le test fonctionnel peut se baser sur un modèle ou une spécification de l'orchestration en termes, par exemple, de processus métier abstraits (e.g. Abstract BPEL [1]), de diagrammes UML [80] ou encore de notation BPMN [111].

Kaschner et al. [112] ont proposé une approche de génération automatique de cas de test pour vérifier la conformité d'une implantation d'un service composé vis à vis de sa spécification. Cette dernière est décrite par des diagrammes d'activités UML, Abstract BPEL ou en notation BPMN. Cette spécification est transformée en un modèle formel : une classe spéciale des réseaux de Pétri, appelée oWFNs, qui a été utilisé dans la génération de cas de test. Cette approche se base sur les partenaires du service composé qui sont supposés interagir avec ce dernier sans *deadlock*. Pour ce test boîte noire, les auteurs ont fait l'hypothèse suivante : l'implantation d'un service composé est conforme à sa spécification si elle n'exclut aucun partenaire, i.e. un service qui communique avec la spécification devrait également communiquer correctement avec l'implantation. Dans ce cas, chaque service partenaire de la spécification a été considéré comme un cas de test de l'implantation.

Frantzen et al. [88] ont proposé une approche formelle pour la modélisation et le test de la coordination de services Web. La spécification de cette coordination de services est fournie en termes de diagrammes de composants UML [80]. Elle décrit certains appels successifs des différentes opérations d'un même service Web. Cette spécification est transformée en un modèle de système de transitions symboliques. Les cas de test sont ensuite dérivés en utilisant les méthodes de test associées à ces systèmes de transitions ainsi que la relation de conformité ioco [113].

Pour le test boîte noire, les domaine des données d'entrée peuvent être infinis ou trop grands pour envisager un test exhaustif, en particulier si les types de données sont complexes. Il est donc nécessaire de limiter les valeurs de ces données afin d'éviter le problème d'explosion du nombre d'états. Le choix de telles valeurs est alors primordial pour dériver un jeu de tests suffisant pour les fonctionnalités à tester. Ce problème se pose aussi pour les services Web et les processus BPEL étant donné qu'ils utilisent et manipulent des données complexes décrites sous forme de schémas XML.

Enfin, nous pouvons noter que le tests à partir d'interfaces WSDL, décrits dans la Section 2.6.1, font partie des tests boîte noire mais sans prise en compte des comportements des services (composés) et de leurs interactions avec leurs services partenaires.

2.7 Quelques autres travaux sur les services Web

Dans cette section, nous allons référencer quelques travaux récents concernant les services Web sémantiques, la vérification et la supervision de la composition de services.

2.7.1 Les services Web sémantiques

Les Web services sémantiques [114] se situent à la convergence des deux domaines : le Web sémantique et les services Web. Le Web sémantique est une représentation abstraite des données du Web pour être facilement traitées par des machines, permettant ainsi plus d'efficacité de découverte, d'automatisation, d'intégration et de réutilisation entre diverses applications. Le Web sémantique est différent du Web qui est généralement syntaxique. Il vise à faciliter la communication homme-machine et machine-à-machine, et la transformation automatique des données. Le Web sémantique et les services Web sont complémentaires. Le premier vise à fournir une interopérabilité sémantique des contenus tandis que les services Web permettent une interopérabilité syntaxique des échanges de données. Les services Web sémantiques vise à créer un Web sémantique de services dont les interfaces et les propriétés sont décrit de manière exploitable par des machines.

Quelques approches [115] ont été développées pour introduire la sémantique aux services Web par l'utilisation des langages suivants : WSDL-S (Web Service Description Language with Semantics) [116], OWL-S language (Ontology Web Language for Services) [117] et WSMO (Web Services Modeling Ontology) [118]. Cette sémantique permet de fournir les informations nécessaires pour l'automatisation des différentes activités liées au cycle de vie d'un service Web : description, publication, composition, supervision, exécution, sélection, etc.

Les services Web sémantiques constituent une voie prometteuse pour mieux exploiter les services Web grâce à l'automatisation de la conception, de la mise œuvre et de la composition de services. Quelques travaux (e.g. [119, 120]) ont proposé des approches de composition automatique de services en vue d'atteindre des fonctionnalités prédéfinies (voir [121] pour plus de détails). Ce type d'approche pourrait constituer une alternance aux langages de composition, en particulier à BPEL. Dans ces contexte, le test pourrait jouer un rôle très important. Pour cela, divers travaux ont essayé, ces dernières années, d'appliquer les techniques de test aux services Web sémantiques (e.g. [122, 123, 124, 125]), et plus particulièrement à la composition automatique des services Web sémantiques (e.g. [126, 127]).

2.7.2 Vérification de l'orchestration de services Web

La conception de la composition de services peut être sujette à des erreurs. Le déploiement de processus BPEL sans aucune vérification peut aboutir à des erreurs d'exécution dont la correction peut être très coûteuse. Il est bénéfique alors d'utiliser des techniques et des outils de vérification pour détecter le comportement incorrect d'un service composé.

[26, 28] ont présenté et analysé différents travaux récents proposant des méthodes formelles pour la vérification et la validation des services composés. Ainsi, de nombreuses approches de vérification ont été proposé (e.g. [33, 39, 128, 129, 130, 131]), incluant les réseaux de Pétri (colorés), les automates et les algèbres de processus. Nous ajoutant à cela, le développement de quelques outils de vérification de BPEL tels que WSAT [57] et WS-Verify [132].

Dans notre travail, nous utiliserons les techniques de vérification pour valider notre transformation d'un processus BPEL en une spécification formelle (en langage IF) qui fera l'objet du Chapitre 4. Cette vérification se fera par le biais des observateurs du langage IF décrivant ainsi les propriétés de l'orchestration de services Web qui doivent être préservées par la spécification IF.

2.7.3 Surpervision des services composés

Différents travaux (e.g. [55, 133, 134, 135, 136, 137]) ont abordé la supervision (ou le montoring) et le diagnostic des services Web et des processus métier implantés en BPEL. Barbon et al [137], pour exemple, ont proposé une approche de supervision des processus BPEL. Cette approche sépare clairement la logique du processus métier de la fonction de supervision. Elle offre la possibilité de surveiller les comportements d'une seule instance d'un processus BPEL ainsi que ceux d'une classe d'instances. Dans cette approche, les contrôleurs peuvent vérifier les propriétés temporelles, statistiques et booléennes. Un langage a été défini pour décrire les spécifications formelles des moniteurs, en plus d'une technique de génération de contrôleurs d'instance ou de classe à partir de leurs spécifications, et de leur traduction automatique en programmes Java.

Yan et al. [138], pour exemple, ont proposé un outil de supervision et de diagnostic des services Web. Cet outil vise à détecter les situations anormales, à identifier les causes de ces anomalies, et à décider les actions de récupération. Pour cela, ils ont défini une méthode automatique de modélisation des comportements d'un processus BPEL et leur interactions par des systèmes à événements discrets, notés DES (cf. 2.5.2). Cette modélisation est une première étape avant de tracer l'évolution du processus métier et de diagnostiquer (ou analyser) ses défauts ou erreurs.

La supervision et le test des services Web composés peuvent être deux tâches complémentaires. Les informations recueillies pendant la supervision d'un service composé peuvent être utilisées pour tester (en ligne) ce service, c'est-à-dire. tester, durant l'exécution du service, si son comportement respecte certaines propriétés.

2.8 Génération de cas de test temporisés

Les systèmes temps réel conjuguent les caractéristiques d'un système réactif soumis aux contraintes imposées par son environnement avec la prise en compte des contraintes temporelles. Ils sont de plus en plus complexes, et leurs exigences en termes de fiabilité et de sûreté de plus en plus forte.

La modélisation formelle des systèmes temps réel ainsi que la génération cas de tests temporisés ont fait l'objet de nombreux travaux (e.g. [139, 140, 141, 142, 143, 144, 145]). Dans ce contexte, de nombreux travaux sont basés sur les automates temporisés [146] et/ou la relation de conformité temporisée notée *tioco*[7] [147]. Nous nous contentons ici de décrire quelques travaux.

Fouchal et al. [139] ont présenté une solution pour réduire la complexité du test des systèmes temporisés. Ils ont proposé un algorithme distribué pour le test de tels systèmes. Mikucionis et al. [145] ont présenté l'outil de test des systèmes embarqués temps réel T-UPPAAL. A partir d'un modèle d'automate temporisé non déterministe, T-UPPAAL permet de dériver et d'exécuter des tests automatiquement en ligne en utilisant le Model-Checker UPPAAL [148]. Cet outil met en œuvre un algorithme de test aléatoire, et utilise une relation de conformité, appelée *relativized timed input/output conformance*, afin d'attribuer un verdict de test.

Berrada et al. [140, 149] ont proposé un modèle générique de systèmes communicants (CS) — supportant des contraintes temporelles et les données — pour traiter différents types et architectures de test tels que le test de conformité, d'interopérabilité et de composants. La sémantique de ce modèle CS a été définie en termes d'automate temporisé à entrée/sortie (ETIOA). Le modèle CS a servi dans la spécification et la génération de cas de test pour le test actif orienté objectifs de test, ou pour le test passif[8]. Berrada et al. ont aussi proposé un algorithme générique de génération de test qui a été mis en œuvre dans l'outil TGSE (Émulation, Simulation et Génération de Test) [149].

2.9 Synthèse

L'objectif de ce chapitre était de définir le cadre dans lequel nos travaux vont se placer, à savoir la composition des services Web, plus précisément, la modélisation et le test de l'orchestration des services Web. Dans ce chapitre, nous avons introduit les services Web, leurs standards, la composition des services et le test de logiciels. Nous avons fait un survol des différents modèles et langages de spécification de la composition de services Web, ainsi que des techniques et approches de test des processus BPEL.

7. *tioco* étend la relation *ioco* de Tretmans et qui est définie comme l'inclusion des traces temporisées entre l'implantation et sa spécification sous la condition que l'implantation doit être réceptive, c.-à-d. elle doit accepter toute entrée dans tout état.

8. Le test passif consiste à vérifier la validité d'une trace d'exécution (exécution valide) de l'implantation sous test.

Dans notre travail, nous nous intéressons, en particulier, au test de conformité de l'orchestration de services Web décrite en BPEL. Nous utiliserons une approche de test boîte grise où les interactions entre le service composé, son client et ses différents partenaires sont connues. Nous nous sommes inspirés des différents modèles proposés pour décrire l'orchestration de services Web afin de proposer une modélisation temporelle de BPEL en termes de machines WS-TEFSM — que nous définirons dans le Chapitre 3. Nous avons aussi pris en considération les différentes approches de test de BPEL, plus précisément, les approches de test boîte noire et blanche, afin de développer notre approche de test boîte grise et décrire une architecture de test. Les travaux de test des systèmes temps réel nous ont servi pour définir notre méthode de génération de tests temporisés — guidée par des objectifs de test. Nous nous sommes aussi inspirés de quelques travaux précédents pour définir, dans le chapitre 5, notre relation de conformité ainsi que notre algorithme de génération de cas de test temporisés — à partir d'une spécification temporelle d'un processus BPEL.

L'objet de la prochaine partie (deuxième partie) de cette thèse est la présentation de notre contribution dans le domaine de la modélisation et le test des services Web orchestrés en BPEL. Cette deuxième partie sera divisée en trois chapitres abordant respectivement, la modélisation temporelle de la composition de services Web, la transformation d'une description BPEL en une spécification formelle en langage IF, et le test des services composés.