Support des propriétés transactionnelles dans les services web*

Nadia Nouali-Taboudjemat, Zohra Mahfoud

Division des théories d'ingénierie et des systèmes d'information, CERIST
Rue des 3 frères Aïssou Ben-Aknoun, Alger, Algérie

onouali@cerist.dz, skirat@cerist.dz, hmeziani@cerist.dz

Résumé : Les dernières décennies ont été marquées par le développement rapide des systèmes d'information distribues, et tout particulièrement par la vulgarisation de l'accès a Internet. Cette évolution a entrainé le développement de nouveaux paradigmes d'interaction entre applications tel que les «services web». La composition de services web permet la création de nouveaux services web (appelés services web composés) par le regroupement des services web existants. Ainsi, les services web composés peuvent être vus comme des applications distribués avec des caractéristiques spécifiques. Ces caractéristiques influencent le traitement transactionnel dans ce domaine et propose une solution flexible pour la composition des services web avec des propriétés transactionnelles. Dans cet article, nous proposons AdapS « Adaptable Composite Web Services » un modèle adaptable et flexible pour la composition de services web avec propriétés transactionnelles. Nous utilisons le formalisme ACTA pour valider et prouver des propriétés d'atomicité du modèle.

Abstract: The latest decades have been marked by the rapid development of the distributed information systems, and particularly by the popularization of Internet access. This evolution led to the development of new paradigms for interaction between applications such as "Web Services". The composition of web services permits to create new web services (called composite web services) by regrouping web services that already exist. Thus, a composite web service can be seen as a distributed application that has specific characteristics. These characteristics influence the transactional aspects in this domain and generate the need for flexible and adaptable solutions for the composition of web services with transactional properties. In this paper, we propose "AdapS" (Adaptable Composite Web Services) a flexible and adaptable model to support the composition of web services with transactional properties. We used the ACTA formalism to validate and prove the atomicity properties of the model.

Mots clés : Composition de services web, modèles transactionnels avancés, propriétés transactionnelles de services web, modèle transactionnel flexible et adaptable pour les services web.

Keywords: Web service composition, Advanced transactional models, Transactional properties of web services, transactional flexible and adaptable model for web services.

* Transactional properties support in web services

1. Les services web et les aspects transactionnels:

Les services Web sont des programmes modulaires, indépendants et auto-descriptifs, qui peuvent être découverts et invoqués via Internet ou un intranet. Les services web présentent une technologie qui permet des interactions entre applications qu'elles soient intra ou interentreprises. Leurs particularités par rapport aux autres technologies de l'informatique répartie résident dans le fait qu'ils offrent un modèle de composants à couplage faible en utilisant la technologie Internet comme infrastructure pour la communication. La technologie des services web utilise des langages et des protocoles qui reposent sur XML[1] (WSDL[2] : pour décrire les services web, SOAP[3] : pour structurer les messages) [Ba08] [Han09].

Un des concepts intéressants qu'offre la technologie des services web est la possibilité de définir un nouveau service à valeur ajouté (dit service web composé) par composition de services web existants. Par exemple, pour partir en vacances, le client fait appel au service composé « organiser un voyage », ce dernier regroupe des services web appartenant à des organisations différentes pour réaliser les fonctionnalités demandées: (acheter un billet de train, réserver une chambre d'hôtel, …).

1.1. Aspects transactionnels dans le domaine des services web

Les services web qui manifestent des propriétés transactionnelles sont ceux qui permettent l'acquisition des ressources par le client. Cette acquisition ne peut s'opérer que par le biais des transactions entre le client et le fournisseur du service. Ainsi, le fournisseur doit doter ses services web d'opérations permettant la cession des ressources. Un service web avec propriétés transactionnelles peut être par exemple un service web permettant la réservation de chambres d'hôtel, de places de spectacle, etc., [Att03][KCQ+08][Han09]. Afin d'illustrer les propriétés transactionnelles des services web composés, nous utilisons la représentation à l'aide du graphe de tâches présentée dans [HS09]. Un service web composé peut être modélisé par un graphe des tâches, où chaque tâche représente une fonctionnalité demandée par le client (figure 1).

[1] eXtensible Markup Language.

[2] Web Services Description Language.

[3] Simple Object Access Protocol.

La figure 1 présente le service web composé SWC « Organiser un voyage » qui comporte quatre tâches t1, t2, t3 et t4 qui représentent respectivement une réservation de chambre de d'hôtel, une réservation de vol, une réservation de train et une réservation de table de

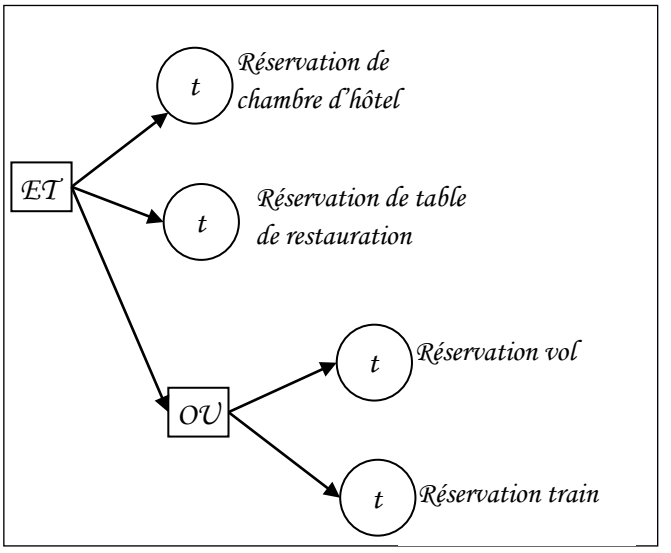

Figure 1: Organisation du voyage par composition de services web

Dans cet exemple, nous remarquons que les réservations ne sont pas d'égale importance, et la composition peut être validée même si la table de restaurant ne peut pas être réservée. Cette tâche est dite non vitale. Alors que la composition ne peut pas être validée si la réservation de chambre d'hôtel a échoué. Cette tâche est dite vitale. Nous remarquons aussi, que bien que la tâche « Réservation de chambre d'hôtel » soit vitale, si lors de la composition du service web, cette tâche ne peut pas être validée, il peut arriver que l'utilisateur accepte de valider le service composé sans qu'il réponde à cette tâche, c'est-à-dire que l'utilisateur peut modifier son avis sur la vitalité d'une tâche lors de l'exécution du service web composé. De plus, si la réservation de vol a échoué (t3 échoue), il est possible de la remplacer par un achat d'un billet de train (t4). Alors, la tâche t3 est dite tâche remplaçable et t2 est sa tâche alternative. La vitalité et la possibilité de remplacement des tâches sont des propriétés transactionnelles qui doivent être exprimées au niveau du service web composé. Ces propriétés peuvent également changer dynamiquement. D'où le besoin de flexibilité dont nous tenons compte dans notre approche.

1.2. Classification transactionnelle des services web élémentaires

Il est possible de caractériser un service web selon les propriétés des transactions qu'il est capable d'exécuter. Plusieurs classifications transactionnelles ont été proposées. Dans [DFDB05, Dua07], une classification basée sur le degré d'atomicité du service web a été proposée où un service web peut être atomique, quasi-atomique ou non-atomique:

- un service est dit d'exécution atomique (c'est-à-dire avec la sémantique du tout ou rien) lorsqu'il offre les opérations suivantes : obtenir (si possible) la réservation d'une ressource, annuler une réservation demandée, valider définitivement l'acquisition d'une ressource réservée ;
- un service quasi-atomique offre une opération pour acquérir (si possible) définitivement une ressource, il offre aussi une opération de compensation ;
- un service est dit non-atomique lorsqu'il fournit seulement une opération pour acquérir définitivement une ressource. Il n'y a pas d'opération de réservation ni de compensation.

1.3. Notions transactionnelles dans la composition de services web

Un service web composé peut être modélisé par un graphe des tâches (voir figure 1). Un service web composé peut être validé même si certaines de ses tâches échouent, ces tâches sont dites non vitales, les autres sont dites vitales. Un service web composé ne peut être validé que si toutes ses tâches vitales ont été validées. Une tâche peut avoir des tâches alternatives ou contingentes; dans ce cas elle est dite remplaçable. Une tâche alternative est exécutée si la tâche à laquelle elle est associée échoue. La vitalité et la « remplaçabilité » des tâches sont des propriétés transactionnelles qui doivent être exprimées au niveau du service web composé. Remarquons que les propriétés transactionnelles d'un service web composés peuvent être changées durant l'exécution. Par exemple si une tâche vitale échoue, l'utilisateur peut rendre cette tâche non vitale empêchant ainsi l'annulation de la composition (exemple de la figure 1).

La sélection des services web participant à une composition consiste à choisir des services web qui répondent aux tâches qui composent un service web composé. Les services web participants sont faiblement couplés, indépendants les uns des autres, et ont des propriétés transactionnelles hétérogènes. Un service web composé doit supporter les propriétés des services web participants à sa composition. En effet, il arrive très fréquemment que plusieurs services web répondent à une même tâche. Ces services sont dits des services alternatifs. Si un service web participant échoue, il est possible de le remplacer par l'un des ses services alternatifs, empêchant ainsi l'abandon de la composition.

Le domaine des services web est dynamique et évolutif ; de nouveaux services peuvent y être ajoutés, les services existants sont constamment modifiés, momentanément suspendus, ou finalement supprimés. Une réponse à cette volatilité des services web peut être le choix dynamiquement (au moment de l'exécution de la composition) des services participants (ou des services web alternatifs), cela permet d'augmenter les chances de validation de la composition.

Il faut noter que les interactions entre les services web ont des durées variables, elles peuvent être de courtes ou de longues durées.

La volatilité des services web, la variabilité et le changement dynamique des propriétés transactionnelles des services web composés rendent difficile, voir

impossible, la prévision statique dès la conception de tous les scénarios pouvant se présenter durant la composition d'un service web composé; d'où le besoin de techniques dynamiquement adaptables.

Les caractéristiques citées ci-dessous montrent le besoin de solutions flexibles et adaptables pour la composition de services web avec propriétés transactionnelles. En effet, plusieurs solutions ont été proposées pour résoudre le problème de la composition des services web avec propriétés transactionnelles. Ces solutions s'inscrivent soit dans le cadre des protocoles spécifiques ou dans le cadre des propositions académiques mais ces solutions présentent, comme nous allons le voir, des limites quant à la prise en charge de la flexibilité et adaptabilité telles que décrites ci-dessus.

1.3.1. Protocoles spécifiques

Les efforts de standardisation ont abouti à un certains nombre de protocoles que nous citons dans ce qui suit :

- **Business Transactions Protocol « BTP »** [Dan03]**:** BTP s'appuie sur une variante du protocole de validation à deux phases pour garantir la propriété d'atomicité. Il introduit deux modèles de transactions: (i) Atom : pour garantir une atomicité stricte. (ii) et Cohésion: qui permet de relâcher la propriété d'atomicité.

  BTP souffre d'un inconvénient majeur : il ne fait pas la distinction entre le protocole de transaction, et les aspects fonctionnels. Aussi, il exige que tous les services web participants supportent les mécanismes de validation à deux phases, cette condition rend la solution très restrictive.

- **Web Services Transaction Framework «WSTF »** [WS-C1.2]: La spécification WSTF sépare le protocole chargé de la coordination « Web Services Coordination: WS-C » du protocole chargé de la transaction « Web Services Transaction WS-T ».

  WS-T introduit deux modèles de transactions: (i) Web Services Atomic Transaction « WS-AT » : garantie l'atomicité stricte pour les transactions courtes. (ii) Web Services Business Activity « WS-BA » : permet le relâchement de la propriété d'atomicité.

  WS-T exige soit que tous les services web participants disposent des mécanismes de validation de deux phases, soit tous les services web participants exposent des opérations de compensation ce qui est très exigeant.

- **Web Services composite application Framework «WS-CAF»** [WS-CTX1.0]**:** Cette spécification sépare le protocole chargé de la gestion de contexte « Web Service Context: WS-CTX », le protocole de coordination « Web Services Coordination: WS-CF » et le protocole chargé de la transaction « Web Services Transaction Manager WS-TXM ».

WS-TXM introduit deux modèles de transactions: (i) ACID Transaction « AT » : garantie l'atomicité stricte, (ii) Long Running Action (LRA) : garantie une atomicité sémantique par l'utilisation des opérations de compensation. (iii) et Business Process (BP): une agrégation des transactions (AT) ou (LRA). « WS-TXM » souffre des mêmes limites que « WS-T ».

1.3.2. Approches académiques

Parmi les solutions académiques proposées, nous avons étudié trois approches que nous avons jugé les plus pertinentes.

– Transactional Composition Of Web Services «TCOWS» [Dua07]:

Le canevas « TCOWS » utilise la classification des services web: atomique, quasi-atomique ou non-atomique.

Le canevas TCOWS comporte trois phases : (i) phase de la conception de la composition avec propriétés transactionnelles, (ii) phase de sélection de participants, (iii) et phase d'exécution de la composition. Le canevas propose de réécrire le module d'orchestration en respectant un ensemble de règles.

Le canevas « TCOWS » présente plusieurs avantages :

- Il permet d'exprimer les propriétés transactionnelles au niveau du service web composé.
- Il respecte les propriétés transactionnelles des services web participants et les utilise lors de la composition.

Mais, il a aussi quelques limites :

- Il suppose que la zone transactionnelle (ensemble des tâches qui doivent exécuter avec la sémantique de tout ou rien) est unique et connexe. Or, la présence de plusieurs zones atomiques non connexes est possible.
- Il ne prend pas en considération le niveau m'imbrication des services web.
- Il ne présente aucune solution si la composition comporte plus d'un service non atomique.

– Web Services In an Agent Based Transaction Model « ABT » [JG03]:

L'approche ABT décrite dans [JG03] définie un système multi-agents pour gérer les transactions pour les services web. Elle propose l'utilisation des quatre agents : (i) l'agent de coordination et de délégation, (ii) l'agent de ressource, (iii) l'agent d'encapsulation et (iv) l'agent de découverte et d'estimation.

Cette approche permet de composer des services web et des ressources locales en :

- respectant l'autonomie des services participants.
- incluant des agents permettant d'encapsuler les services web. Ces derniers permettent de cacher l'hétérogénéité des services web participants.
- incluant un agent de découvert et d'estimation évolutif.
- utilisant une liste des participants alternatifs.

Mais cette approche n'a pas bien étudié les problèmes transactionnels rencontrés lors de la composition de services web à savoir :

- l'expression des propriétés transactionnelles au niveau de service web composé n'est pas étudiée.
- les interactions entre les agents d'encapsulation et les services encapsulés.

– **Reliable Web Services Compositions By Ensuring The Failure Atomicity** [Bhi05]:

Cette étude propose un modèle de services web transactionnel qui a pour but d'assurer des exécutions correctes des services web composés. Ce modèle distingue en particulier entre le flot de contrôle et le flot transactionnel d'un service Web composé transactionnel (SCT). Le flot de contrôle définit l'ordre d'invocation des services composants. Le flot transactionnel définit les mécanismes de recouvrement en cas d'échecs.

Dans ce modèle, trois approches ont été développées pour assurer des compositions fiables de services Web : la première repose sur la validation du modèle de composition du SCT conformément aux besoins des concepteurs. La deuxième procède par réingénierie du SCT. La troisième repose sur le concept de patron transactionnel.

Bien que cette approche présente une solution pour assurer des compositions fiable des services web, elle ne permet pas d'exprimer les propriétés transactionnelles au niveau du service web composé.

1.4. Comparaison

Le tableau 1 présente une comparaison des différentes solutions étudiées ci-dessus, selon les deux critères : flexibilité et adaptabilité.

Critères de comparaison		Solutions	Protocoles			Approches Académiques		
			BTP	WSTF	WS-CAF	TCOW	ABT	[Bhi05]
Flexibilité	Exprimer et exploiter les aspects transactionnels au niveau des services web composés.	Vitalité des tâches	+	+	+	+	-	-
		Remplaçabilité des tâches	+	+	-	+	-	+
	Supporter les différents types et propriétés des services web	SW Compensable	+	+	+	+	+	+
		SW Quasi-Compensable	+	+	+	+	+	-
		SW Pivot	-	-	-	+/-	-	+
		SW Re-exécutable (rejouable)	-	-	+	-	-	+
		SW Alternatif	+	+	+	+	+	+
	Respecter l'autonomie des services web participants		+	-	+	+	+	+
Adaptabilité	Choix et remplacement dynamique des services web participants		-	-	-	+	-	-
	Adaptabilité au changement dynamique des besoins transactionnels des utilisateurs		-	+	-	-	-	-

Tableau 1: Comparaison entre les différentes solutions qui s'intéressent aux aspects transactionnels dans les services web.

L'analyse du tableau 1, nous a permis de constater que les différentes solutions étudiées :

- ne supportent pas les services web pivots : Comme ces derniers n'offrent pas des mécanismes de recouvrement, son invocation peut entraîner la violence de la cohérence du système,
- n'exploitent pas la possibilité de ré-exécuter les services web participants, or que cette propriété augmente les chances de validation des compositions des services web,
- ne profitent pas des bénéfices du choix et de remplacement dynamique des services web participants,
- ne s'adaptent pas au changement dynamique des besoins transactionnels des clients.

Et ainsi, les solutions proposées ne répondent pas parfaitement aux besoins transactionnels dans le domaine des services web et surtout en termes d'adaptabilité.

2. Modèle adaptable pour les services web composés « *Adaptable Composite Web Services -AdapS-* »

Nous présentons dans cette section le modèle *« AdapS »* [4] qui a pour but d'offrir un support flexible et adaptable pour la composition de services web avec propriétés transactionnelles.

2.1. Présentation du modèle « *AdapS* »

Nous proposons de définir un service web composé comme un ensemble des tâches, chaque tâche peut être exécutée par des services web équivalents (qui offrent les mêmes fonctionnalités, la même capacité de service) [HS09].

Alors, nous modélisons un service web composé adaptable « *AdapS* » par un arbre dont les nœuds internes sont les tâches et les feuilles sont les services web désignés pour les exécuter (figure2).

- Les tâches peuvent être : (vitales/ non vitales), (remplaçable/ non remplaçable).
- Les services web participants peuvent être : composés ou élémentaires.
- Un service web composé participant est modélisé à son tour par un arbre des tâches et des services web.
- Un service web participant peut être : (compensable, quasi compensable ou pivot), (ré-exécutable ou non ré-exécutable), (remplaçable ou non remplaçable).
- Un plan d'exécution est le résultat d'affectation d'un seul service web à chaque tâche.

Un service est :

- **Compensable :** s'il offre une opération pour acquérir (si possible) définitivement une ressource, il offre aussi une opération de compensation[5], les opérations de compensation sont généralement liées aux coûts de compensation (ce type est dit aussi quasi-atomique).

[4] Ce modèle nous a été inspiré par les travaux de [Nou07]

[5] Une opération de compensation a pour objectif d'annuler les effets d'une autre opération qui n'a pas pu être terminée avec succès.

- **Quasi-compensable :** s'il offre les opérations suivantes : (i) obtenir (si possible) la réservation d'une ressource pour une période limitée (Timeout), (ii) annuler une réservation demandée, (iii) valider définitivement l'acquisition d'une ressource réservée (ce type est dit aussi atomique).
- **Pivot :** s'il fournit seulement une opération pour acquérir définitivement une ressource. Il n'y a pas d'opération de réservation ni de compensation (ce type est dit aussi non-atomique).
- **Ré-exécutable :** En cas d'échec, la réexécution d'un service ré-exécutable peut conduire à sa validation. Un service ré-exécutable est caractérisé par un nombre fini de tentatives permis, et un délai d'attente entre les tentatives d'exécution.

Notons qu'un service web peut être en même temps (Compensable et ré-exécutable), (Quasi-compensable et ré-exécutable) ou (pivot et ré-exécutable).

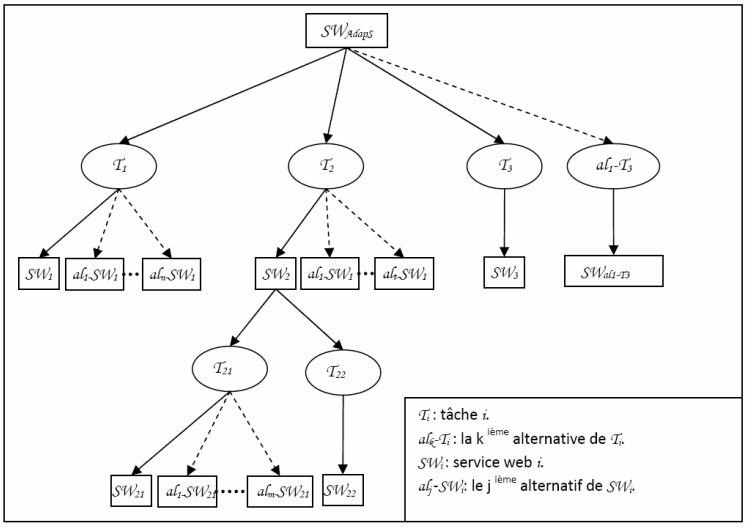

Figure 2: Structure d'arbre d'un service web composé adaptable « AdapS ».

2.2. Définition du modèle « *AdapS* »

Nous modélisons un service web composé adaptable comme suit : $\mathcal{SW}_{AdapS} = \{(\mathcal{T}_i, \mathcal{RT}_i, \mathcal{LSw}_{Ti}, al\text{-}\mathcal{T}_i, i \geq 1)\}$

avec :

- $\mathcal{T}_i$ est une tâche du service web composé adaptable $\mathcal{SW}$.
- $\mathcal{RT}_i = (r_1, r_2)$ spécifie le rôle de la tâche $\mathcal{T}_i$.

$r_1 \in \{Vt, Nvt\}$, $r_2 \in \{Rp, Nrp\}$.

Vt : tâche vitale, Nvt : tâche non vitale,

Rp : tâche remplaçable, Nrp: tâche non remplaçable.

- $LSw_{Ti} = (Sw_i, RSw_i, al\text{-}Sw_i)$: présente la liste des services web qui peuvent exécuter la tâche i :
  - Sw_i : est l'identité de service web principal choisit pour exécuter la tâche i.
  - RSw_i : spécifie le rôle du service web principale Sw_i : $RSw_i = (rs_1, rs_2, rs_3)$.

    Avec : $rs_1 \in \{Cp, QCp, P, \varnothing\}$, $rs_2 \in \{Rx, Nrx, \varnothing\}$, $rs_3 \in \{Rp, Nrp\}$.

    Cp: compensable, QCp : quasi-compensable , P: pivot,

    Rx: ré-exécutable, Nrx: non ré-exécutable,

    Rp : remplaçable, Nrp: non remplaçable.

    $\varnothing$: la propriété n'est pas pertinente (le service est composé).
- *al-Swi = {(alk-Swi,Ralk-Swi), k≥0} : présente la liste des alternatifs du service principale Swi, avec:*
  - *alk- Swi : le k ième alternatif du service principal Swi,.*
  - *Ralk-Swi : spécifie le rôle du service alternatif alk-Swi. Par le même principe utilisé précédemment, nous décrivons le rôle Ralk-Swi= (rs1, rs2) avec :*

    $rs1 \in \{Cp, QCp, P, \varnothing\}$, $rs2 \in \{Rx, Nrx, \varnothing\}$.
  - *al-Ti = {(alk-Ti , LSwalk-Ti, k≥0,) } représente la liste des tâches alternatives qui peuvent substituer la tâche principale Ti , et leurs sites d'exécutions, avec:*

    alk-Ti : la kième alternative de la tâche Ti.

    LSwalk-Ti présente la liste des services web qui peuvent exécuter l'alternative alk-Ti. La structure de LSwalk-Ti est similaire à celle de LSwTi définit ci-dessus.

Il existe une relation de dépendance RD entre les tâches du même service web composé :

$\forall (Ti, RTi, LSwTi, al\text{-}Ti), (Tj, RTj, LSwTj, al\text{-}Tj) \in SWAMapS$

$(Ti, RTi, LSwTi, al\text{-}Ti)\ RD\ (Tj, RTj, LSwTj, al\text{-}Tj)$

RD est une relation de dépendance de parallélisme ($\|$) ou de séquentialité (<).

2.3. Adaptabilité du modèle « *AdapS* »

Afin d'offrir la flexibilité et l'adaptabilité à un service web composé, nous proposons que :

- *à la phase de conception :* seules les tâches composantes le service adaptable et les relations de dépendance entre elles sont définies,
- *à la phase d'exécution:* L'utilisateur décide quelles sont les tâches vitales, les tâches non-vitales, les tâches remplaçables, et leurs tâches alternatives. Par la suite, le service web composé décide la liste des services participants principaux ainsi que la liste des services alternatifs. Et après, il invoque les services web principaux à exécuter les tâches associées.

Si pour une raison ou une autre un service web ne peut pas valider la tâche associée, la suite de l'exécution sera décidée selon le rôle de ce service et de la tâche appropriée:

- Le service est ré-exécutable, retenter l'exécution de la tâche.
- Le service est remplaçable, lancer l'un des services web alternatifs.
- La tâche est remplaçable, remplacer la tâche par l'une des tâches alternatives.
- La tâche n'est pas vitale, la tâche défaillante est juste ignorée et la composition de service adaptable poursuit son exécution (un service composé peut être validé même si l'une des tâches non vitales a échoué).
- La tâche est vitale, non remplaçable, non ré-exécutable, aucun service associé n'a achevé son exécution, la composition de service web adaptable doit être abandonnée.

Afin d'augmenter les chances de validation des services web composés adaptables, nous proposons d'offrir la possibilité de modifier les rôles des tâches et des services web associés en cours de l'exécution, comme suit :

- Retenter l'exécution d'un service même si au départ il n'était pas signalé comme ré-exécutable. Par exemple, le service web offrant la réservation d'un taxi peut être validé s'il est retenté plusieurs fois, même s'il n'était pas définie comme ré-exécutable lors de sa conception.
- Mettre à jour la liste des services web alternatifs.
- Rendre la tâche non vitale (l'annulation d'une tâche non vitale n'influence pas le processus de la composition).

Le modèle *AdapS* offre un support pour garantir une exécution qui respecte différents niveaux d'atomicité (atomicité stricte, sémantique ou relâchée) [6]. Il permet Aussi, de s'adapter de façon dynamique à la variation du degré d'atomicité durant la composition d'un service web adaptable. Un service web adaptable peut être d'une:

Atomicité Stricte :

L'atomicité stricte, fait référence au fait que tous les services web participants valident les tâches demandées ou aucun service n'est validé, c'est l'atomicité standard « tout ou rien ».

Dans le cadre de l'atomicité stricte, toutes les tâches sont vitales non remplaçables, les services web participants sont tous non remplaçables et non ré-exécutables, au plus un service pivot et les autres sont quasi-compensables.

Pour valider la composition d'un service adaptable avec atomicité stricte, les ressources offertes par les services quasi-compensables doivent être toutes réservées avant l'invocation du service pivot (s'il existe), de façon que si les ressources exposées par ce dernier sont acquises la composition sera validée ainsi que toutes les réservations effectuées par les services quasi-compensables. Dans le cas contraire (un ou des services web ne peuvent pas valider les tâches demandées), la composition sera annulée ainsi que toutes les réservations qui ont été effectuées.

Prenons l'exemple du service web d'« organisation d'une réunion » composé de trois services web: « réservation d'une salle », « réservation du traiteur » et « invitation des participants », ce service est caractérisé par un respect strict de la contrainte d'atomicité, si l'un des services web participants échoue la réunion doit être annulée.

Atomicité Sémantique :

L'atomicité sémantique est caractérisée par la présence des services web compensables. Pour valider la composition d'un service adaptable avec atomicité sémantique, les ressources offertes par les services quasi-compensables/compensables doivent être toutes réservées/acquises avant l'invocation du service pivot (s'il existe), de façon que si les ressources exposées par ce dernier sont acquises, la composition sera validée ainsi que toutes les

[6] Il faut noter qu'un plan d'exécution ne peut pas contenir plus d'un service web pivot associé à une tâche vitale. Dans le cas contraire les contraintes d'atomicité du système peuvent être violées. Ceci est discuté dans la partie consacrée au protocole « aWST ».

réservations effectuées par les services quasi-compensables. Dans le cas, contraire (un ou des services web ne peuvent pas valider les tâches demandées), la composition sera abandonnée et les services quasi-compensables/compensables procèdent à des annulations/compensations.

Supposons que dans l'exemple d'organiser un voyage, le service web qui assure l'achat de billet de train est compensable, lors de son invocation le billet sera automatiquement acheté. Si par la suite la composition sera annulée, on lance l'opération de compensation qui consiste par exemple à récupérer un pourcentage du prix payé.

Atomicité Relâchée:

L'atomicité est relâchée dans le sens où l'échec de certains services web ou tâches n'implique pas obligatoirement l'abandon de la composition. Ce type d'atomicité est obtenue lorsque le service composé consiste en une combinaison quelconque de tâches vitales ou non vitales, remplaçables ou non, des services web participants compensables, quasi-compensables ou pivots, remplaçables ou non, ré-exécutable ou non. Dans ce cadre, il est possible de retenter l'exécution d'un service web qui a échoué. En plus, la composition du service adaptable peut être achevée même si une ou plusieurs tâches non vitales ont été échouées.

Notons ici que même si l'atomicité est relâchée, il faut que la composition respecte certaines contraintes d'atomicité afin de préserver la cohérence du système :

- Une composition ne peut être validée que si et seulement si toutes les tâches vitales sont validées.
- Si une composition doit être annulée, elle doit apparaître comme si elle n'a été jamais exécutée, tous les services web participants doivent annuler le travail effectué soit d'une façon réelle ou d'une façon sémantique.

Pour valider la composition d'un service adaptable avec atomicité relachée, les ressources offertes par les services quasi-compensables/compensables associés aux tâches vitales doivent être toutes réservées/acquises avant l'invocation du service pivot associé à une tâche vitale (s'il existe), de façon que si les ressources exposées par ce dernier sont acquises, la composition sera validée ainsi que toutes les réservations effectuées par les services quasi-compensables. Dans le cas contraire (un ou des services web ne peuvent pas valider les tâches demandées), la composition sera abandonnée et les services quasi-compensables/compensables procèdent à des annulations/compensations. Aussi, les ressources offertes par les services pivots associés aux tâches non vitales ne sont acquises qu'après que le service adaptable décide de valider la composition.

Par exemple, le service web qui permet d'« organiser un voyage », peut valider même si la réservation de la table de restaurant n'est pas réalisée, ce qui implique le relâchement de la propriété d'atomicité.

Remarquons que l'introduction de la possibilité de changement des rôles en cours d'exécution, permet au modèle AdapS de s'adapter de façon dynamique à la variation du degré d'atomicité durant la composition d'un service web adaptable.

Exemple :

Nous prenons l'exemple du service web SW « Organiser un voyage » qui permet d'acheter un billet d'avion, et si ce n'est pas possible d'acheter un billet de train, de réserver une chambre d'hôtel, et éventuellement de contacter un guide touristique. Le vol et la chambre sont vitaux.

La liste des tâches est définie comme suit :

- $\mathcal{T}_1$ *:* réserver un vol de la ville A vers la ville B.
- al_1-$\mathcal{T}_1$ *:* acheter un billet de train de la ville A vers la ville B.
- $\mathcal{T}_2$: réserver une chambre d'hôtel.
- $\mathcal{T}_3$ *:* contacter un guide touristique.

Lors de la sélection des services web participants à la composition du service adaptable $\mathcal{SW}$:

- le service web $\mathcal{S}w_1$ est désigné pour exécuter $\mathcal{T}_1$: supposons qu'il n'est pas possible de réserver un vol direct de la ville A vers la ville B, $\mathcal{S}w_1$ compose deux services web pour satisfaire la tâche $\mathcal{T}_1$. Donc $\mathcal{S}w_1$ est un service web composé des deux tâches :
- $\mathcal{T}_{11}$: Réserver un vol de la ville A vers la ville X, exécutée par le service $\mathcal{S}w_{11}$.
- $\mathcal{T}_{12}$: Réserver un vol de la ville X vers la ville B, exécutée par le service $\mathcal{S}w_{12}$.
- ($\mathcal{S}w_{11}$: est quasi-compensable et re-exécutable, $\mathcal{S}w_{12}$: est compensable).
- le service web $\mathcal{S}w_1$' est désigné pour exécuter la tâche alternative al_1-$\mathcal{T}_1$ ($\mathcal{S}w_1$'est quasi-compensable).
- Le service web $\mathcal{S}w_2$ est désigné pour exécuter la tâche $\mathcal{T}_2$, le service web al_1-$\mathcal{S}w_2$ est son alternatif ($\mathcal{S}w_2$: quasi-compensable et re-exécutable, al_1-$\mathcal{S}w_2$: compensable).
- le service web $\mathcal{S}w_3$ est sélectionné pour exécuter la tâche $\mathcal{T}_3$.

La structure du service composé adaptable $\mathcal{SW}$ est donnée par la figure 3 :

Ainsi, le service web Sw est décrit comme suit :

$$Sw_{AdapS}=\{(T_1,(Vt,Rp),(Sw_1,(\emptyset,\emptyset,Nrp),\emptyset),\{al_1\text{-}T_1,(Sw_1',(Cp,Nrx,Nrp),\emptyset)\}),(T_2,(Vt,Nrp),(Sw_2,(P,Rx,Rp),\{al_1\text{-}Sw_2(P,Nrx)\}),\emptyset),(T_3,(Nvt,Nrp),(Sw_3,(Qcp,Nrx,Nrp),\emptyset),\emptyset)\}.$$

Et :

$$Sw_{1AdapS}=\{(T_{11},(Vt,Nrp),(Sw_{11},(Qcp,Rx,Nrp),\emptyset),\emptyset),(T_{12},(Vt,Nrp),(Sw_{12},(Cp,Nrx,Nrp),\emptyset),\emptyset)\}$$

Lors de la composition du service adaptable « SW », si par exemple Sw_{11} échoue et la réservation du billet de vol entre la ville A et X ne peut pas être réalisée, le modèle donne la possibilité à l'utilisateur de changer d'avis sur la vitalité de la tâche T_{11}. Le client peut accepter de s'occuper de son déplacement de la ville A vers la vile X, et demande au « SW » de règler les autres tâches.

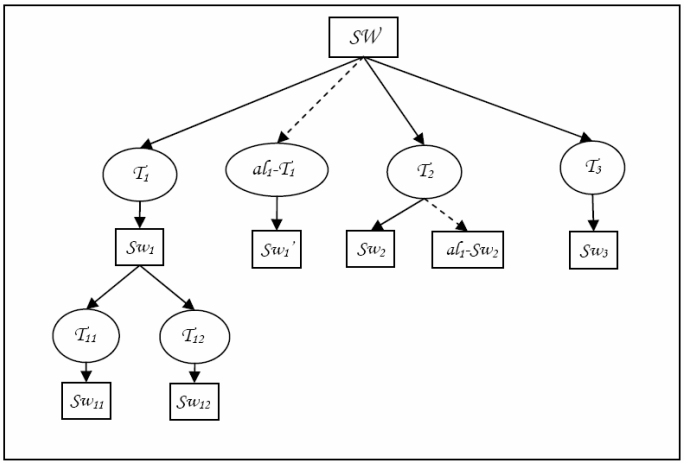

Figure 3: Structure générale du SW « Organiser un voyage ».

3. Spécification formelle du modèle « AdapS »

Nous proposons de spécifier le modèle « $AdapS$ » en utilisant le formalisme ACTA [CR91, CR94, Nou07]. ACTA est un framework développé pour définir les modèles de transactions et démontrer leurs propriétés. Ce formalisme permet de spécifier : (1) les effets des transactions sur d'autres transactions et (2) les effets des transactions sur les objets.

Afin de spécifier le modèle « $AdapS$ » par le formalisme ACTA nous devons transformer le modèle « $AdapS$ » en une représentation basée sur les transactions. En effet, un service web composé adaptable peut être considéré comme une transaction composée de plusieurs sous-transactions. Les sous-transactions sont exécutées par les services web participants, leurs propriétés sont donc issues des

propriétés transactionnelles des services web participants, et aussi des tâches auxquelles ces services sont associés.

Un service web composé selon le modèle *« AdapS »* est une transaction adaptable composée des sous-transactions. Une sous-transaction peut être décomposée en n'importe quel niveau d'emboîtement, qui donne une transaction racine et un ensemble de sous-transactions (ou transactions composantes). Ces transactions s'appellent transactions principales. Des transactions secondaires, compensatrices et alternatives, peuvent être associées aux sous-transactions composantes principales. Les transactions de compensation ou compensatrices peuvent également être associées à des transactions alternatives. Les transactions peuvent avoir plusieurs rôles ; elles peuvent être compensables/quasi-compensables ou pivots, vitales/non-vitales, re-exécutables/non re-exécutables, et remplaçables/non-remplaçables.

Dans ce qui suit nous décrivons les dépendances entre les sous-transactions et la racine (transactions principales) et entre les sous-transactions principales et les sous-transactions secondaires. Les notations utilisées sont les suivantes:

- *T* est un service web composé adaptable du modèle *« AdapS »*. Alors, il est considéré comme une transaction adaptable du modèle *« AdapS »*, $T \neq T.'$
- $T = \{T_1 (r_1, r_2, r_3, r_4), \ldots, T_n (r_1, r_2, r_3, r_4)\}, n > 0$

  r1 ∈{Cp, QCp, P}, r2∈{Rp, NRp}, r3∈{Rx, NRx}, r4∈{Vt, NVt}.

  Avec Cp : compensable, QCp : quasi-compensable, P : pivot.

  Rp : remplaçable, NRp : non-remplaçable.

  Rx : réexécutable, NRx : non-réexécutable.

  Vt : vitale, NVt : non-vitale.

- *Comp-Trs* est l'ensemble des sous-transactions compensables, *QComp-Trs* est l'ensemble des sous-transactions quasi-compensables et *Pivot-Trs* est l'ensemble des sous-transactions pivots : $Comp\text{-}Trs \cap QComp\text{-}Trs \cap Pivot\text{-}Trs = \varnothing$.
- *Vt-Trs* présente l'ensemble des sous-transactions vitales. Elle comporte l'ensemble des sous-transactions quasi-compensables vitales (*QCpVt-Trs)*, l'ensemble des sous-transactions quasi-compensables (*CpVt-Trs)*, et la seule transaction pivot vitale T_i(P,r2,r3,Vt): $Vt\text{-}Trs = QCpVt\text{-}Trs \cup CpVt\text{-}Trs \cup T_i(P,r2,r3,Vt)$.
- $Comp\text{-}T_i$ est la sous-transaction compensatrice de T_i.
- $alt\text{-}T_i = \{aT_{i1} (r1, r3), \ldots, aT_{is} (r1, r3)\}$, $s \geq 0$ est l'ensemble des sous-transactions alternatives d'une T_i *avec : r1 ∈{Cp, QCp, P}, r3∈{Rx, NRx}*.

  Avec, Cp : compensable, QCp : quasi-compensable, P : pivot.

  Rx : réexécutable, NRx : non-réexécutable.

– *t dénote Ti ou comp-Ti ou alt-Tis.*

– *Ti (r1, r2, r3, r4) dénote une sous-transaction principale i avec son rôle.*

– *Tp dénote une transaction « AdapS » parent* {*racine ou autre*}.

– *Su dénote l'ensemble des transactions qui s'exécutent sur le service web SW.*

– *H dénote l'historique de l'exécution de la transaction T*[7].

3.1. Définition axiomatique du modèle « AdapS »

(1) $ES_T = ES_t$ = {begin, commit, prepare, abort}
(2) EI_T , EI_t = {begin}
(3) ET_T , ET_t = {commit, abort}
(4) t satisfait les axiomes fondamentaux I à IV [CR90].

(5) $(begin\ t_i \in H) \Rightarrow ((t_i\ BD\ T) \in DepSet_{ct}$[8]).
Le début d'une sous-transaction *t* est conditionné par le démarrage de la transaction racine T.

(6) $T_i\ (r1, r2, r3, r4)\ BCD\ T_j\ (r1, r2, r3, r4)$
$(begin_{Ti} \in H) \Rightarrow (commit_{Tj} \rightarrow begin_{Ti})$
Cet axiome établie un ordre partiel entre les sous-transactions inter dépendantes.

(7) $T_i\ (Cp, r2, r3, r4)\ WD\ T_P$
$(abort_{Tp} \in H) \Rightarrow (\neg\ (commit_{Ti} \rightarrow abort_{Tp}) \Rightarrow (abort_{Ti} \in H))$
Cet axiome garanti l'annulation d'une sous-transaction enfant compensable qui n'a pas encore validé en cas d'annulation du parent.

(8) T_i(QCp, *r2, r3, r4) CD* T_P
$(commit_{Ti} \in H) \Rightarrow ((commit_{Tp} \in H) \Rightarrow (commit_{Tp} \rightarrow commit_{Ti}))$

T_i(QCp, *r2, r3, r4) AD* T_P
$(abort_{Tp} \in H) \Rightarrow (abort_{Ti} \in H)$
Cet axiome oblige les sous-transactions quasi-compensables à ne jamais valider avant leurs parents. Si le parent annule alors les enfants quasi-compensables annulent.

---

[7] Un historique H de l'exécution concurrente d'un ensemble de transactions T, contient tous les événements associés à T et indique l'ordre (partiel) de l'occurrence de ces événements. H_{ct} dénote l'historique des événements produits avant un instant donné (historique courant).

[8] *DepSetct* dénote l'ensemble de dépendances jusqu'à un instant donné (ct).

(9) T_p *CD* T_i *(Cp, r2, r3, Vt) &* T_P *CPD* T_j *(QCp, r2, r3, Vt) &* T_P *SCD* T_k *(P, r2, r3, Vt)*
$(commit_{TP} \in H) \Rightarrow ((commit_{Ti} \in H) \Rightarrow (commit_{Ti} \rightarrow commit_{Tp})) \wedge (Prepare_{Tj} \rightarrow commit_{Tp}) \wedge ((commit_{Tk} \in H) \Rightarrow (commit_{Tp}))$.
La validation de la transaction parent dépend de :
- la validation de toutes les sous-transactions compensables, et de
- la préparation de toutes les sous transactions quasi-compensables vitales.

Aussi, la validation de la transaction pivot vitale implique la validation du parent.

(10) T_i *(P, r2, r3, Vt) CD* T_j *(Cp, r2, r3, Vt) &* T_i *(P, r2, r3, Vt) CPD* T_k *(QCp, r2, r3, Vt)*
$(commit_{Ti} \in H) \Rightarrow ((commit_{Tj} \in H) \Rightarrow (commit_{Tj} \rightarrow commit_{Ti})) \wedge (prepare_{Tk} \rightarrow commit_{Ti}))$.
Cet axiome oblige la sous-transaction pivot vitale à ne pas valider qu'après :
- la validation des toutes les sous-transactions compensables vitales et
- la préparation des toutes les sous-transactions quasi-compensables vitales.

(11) T_i *(QCp, r2, r3, Vt) SCD* T_P
$(commit_{Tp} \in H) \Rightarrow (commit_{Ti}))$.
La validation de la transaction parent implique la validation de toutes les transactions composantes quasi-compensables vitales.

(12) T_i *(P, r2, r3,NVt) BCD* T_p
$(commit_{Ti} \in H) \Rightarrow ((commit_{Tp}) \Rightarrow (commit_{Tp} \rightarrow commit_{Ti})$
Cet axiome implique le lancement des sous-transactions pivots non-vitales après la validation de leur transaction parent.

(13) $Comp\text{-}T_i(r_1, r_2, r_3, r_4)$ *BCD* T_i *(Cp,* r_2, r_3, r_4*) &* $Comp\text{-}T_i(r_1, r_2, r_3, r_4)$ *BAD* T_P.
$(begin_{comp\text{-}Ti} \in H) \Rightarrow (commit_{Tj} \rightarrow begin_{comp\text{-}Ti})$ & $(begin_{comp\text{-}Ti} \in H) \Rightarrow (abort_{Tp} \rightarrow begin_{comp\text{-}Ti})$.
Une transaction compensatrice d'une transaction ne peut pas commencer leur exécution avant la validation de cette dernière. De plus, la transaction compensatrice ne peut pas démarrer avant l'annulation de la transaction parent.

$Comp\text{-}T_i(r1, r2, r3, r4)$ *CMD* T_P
$(abort_{Tp} \in H) \Rightarrow (commit_{comp\text{-}Ti} \in H)$
Pour les sous-transactions compensables qui ont validées, si leurs parents annulent alors les sous-transactions compensatrices doivent valider.

(14) $alt\text{-}T_{is}$ = sième alternative de T_i
$alt\text{-}T_{is}$ (r_1, r_3) *BAD* T_i (r_1, Rp, r_3, r_4)
$(begin\ alt\text{-}T_{is} \in H) \Rightarrow (abort_{Ti} \rightarrow begin_{alt\text{-}Tis})$
Une transaction alternative ne peut commencer avant l'annulation de la transaction à laquelle elle est associée.

$Alt\text{-}T_{is}(r_1, r_3)$ *BAD* $Alt\text{-}T_{is\text{-}1}$ (r_1, r_3).
$(begin\ alt\text{-}T_{is} \in H) \Rightarrow (abort\ alt\text{-}T_{js\text{-}1} \rightarrow begin\ alt\text{-}T_{is})$
Une transaction alternative ne peut commencer avant l'annulation d'une transaction alternative précédente.

(15) T_P *AD* T_i (r_1, r_2, r_3, Vt)
$(abort_{Ti} \in H) \Rightarrow (abort_{Tp} \in H)$
Si une sous-transaction vitale annule, alors la transaction parent doit annuler.

(16) $T_i(r1, r2, Rx, r4)$ *BAD* T_i $(r1, r2, Rx, r4)$
$(begin_{Ti} \in H) \Rightarrow (abort_{Ti} \rightarrow begin_{Ti})$
Une sous-transaction ne peut être re-exécutée que si la tentative précédente a été annulée.

Les axiomes de (1) à (3) de la définition axiomatique indiquent les événements significatifs du modèle « AdapS ». L'axiome (4) dit que les sous-transactions ainsi que leurs transactions de compensation et alternatives doivent respecter les axiomes fondamentaux[9]. L'axiome (5) dit que l'exécution des sous-transactions démarre après le démarrage de la transaction principale. Les axiomes de (6) à (16) expriment les différentes dépendances entre les transactions. Ces dépendances spécifient les liens qui découlent directement de la structure. Par exemple, qu'un ordre partiel existe entre les sous-transactions qui sont liées (axiome6). Aussi, une transaction pivot vitale ne peut être validée qu'après la validation de toutes les sous-transactions compensables vitales et la préparation de toutes les sous-transactions quasi-compensables (axiome 10), etc.

Remarque *:*

Les dépendances utilisées sont les suivantes :

- **Dépendance de validation (Commit Dependency)** (t_j *CD* t_i)**:** si les deux transactions t_i et t_j valident, la validation de t_i doit précéder celle de t_j : $(commit_{tj} \in H) \Rightarrow ((commit_{ti} \in H) \Rightarrow (commit_{ti} \rightarrow commit_{tj}))$.
- **Dépendance de validation forte (Strong-Commit Dependency)** $(t_j\ SC\ t_i)$ *: si* t_i *valide, alors* t_j *valide aussi:* $(commit_{tj} \in H) \Rightarrow (commit_{ti} \in H)$.
- **Dépendance d'annulation (Abort Dependency)** (t_j *AD* t_i) : si t_i annule, alors t_j aussi: $(abort_{ti} \in H)) \Rightarrow (abort_{tj} \in H)$.

[9] I. Une transaction ne peut pas être initiée par deux événements différents.

II. Si une transaction est terminée, elle a dû auparavant être initiée.

III. Une transaction ne peut pas être terminée par deux événements différents.

IV. Seules les transactions en cours peuvent demander des opérations sur les objets. Cet axiome énonce simultanément qu'une transaction doit toujours être initiée et terminée.

- **Dépendance d'annulation faible (Weak-Abort Dependency)** (t_j *WD* t_i): si t_i annule et t_j n'a pas encore validé, t_j doit annuler aussi. En d'autres termes, si t_j valide et t_i annule, la validation de t_j doit précéder l'annulation de t_i dans l'historique: $(abort_{ti} \in H) \Rightarrow (\neg\ (commit_{tj} \rightarrow abort_{ti}) \Rightarrow (abort_{tj} \in H))$.
- **Dépendance de compensation (Force-Commit-on-Abort Dependency)** (t_j *CMD* t_i): si t_i annule, t_j doit valider: $(abort_{ti} \in H) \Rightarrow (commit_{tj} \in H)$.
- **Dépendance de début (Begin Dependency)** (t_j *BD* t_i): t_j ne peut pas démarrer si t_i n'a pas démarré: $(begin_{tj} \in H) \Rightarrow (begin_{ti} \rightarrow begin_{tj})$.
- **Dépendance début sur validation (Begin-on-Commit Dependency)** (t_j *BCD* t_i): t_j ne peut pas démarrer jusqu'à ce que t_i valide: $(begin_{tj} \in H) \Rightarrow (commit_{ti} \rightarrow begin_{tj})$.
- **- Dépendance début sur annulation (Begin-on-Abort Dependency)** (t_j *BAD* t_i): t_j ne peut pas démarrer jusqu'à ce que t_i annule: $(begin_{tj} \in H) \Rightarrow (abort_{ti} \rightarrow begin_{tj})$.

  Remarquons qu'en plus des dépendances définis par le formalisme ACTA, nous avons défini la dépendance de validation sur préparation *CPD* comme suit :

- **- Dépendance validation sur préparation (Commit-on-Prepare Dependency)** (t_j *CPD* t_i): t_j ne peut pas valider jusqu'à ce que t_i prepare: $(commit_{tj} \in H) \Rightarrow (prepare_{ti} \rightarrow commit_{tj})$.

3.2. Les propriétés d'atomicité du modèle «AdapS »

Dans cette section nous démontrons à l'aide du formalisme "ACTA" qu'un service web «AdapS » peut avoir un niveau d'atomicité stricte, sémantique ou relâché.

3.2.1. Atomicité stricte d'une transaction *« AdapS »*

Pour montrer que les transactions « *AdapS* » peuvent avoir la propriété d'atomicité stricte, nous commençons par définir la propriété d'atomicité stricte (définition 02). Puis, nous décrivons les conditions pour qu'une transaction T_{st} du modèle « *AdapS* » assure une atomicité stricte (Lemme 1). Ensuite, nous spécifions la validation de la transaction T_{st} (Lemme 1.1) ainsi que son annulation (Lemme 1.2). Et enfin, nous déduisons l'atomicité stricte de la transaction T_{st}.

Définition 02 : Atomicité stricte.

Une transaction T assure l'atomicité stricte:

si T est validée, toutes les transactions composantes Ti doivent aussi être validées, et

si T est annulée, toutes les transactions Ti validées doivent être annulées.

Autrement dit:

(commit$_T$ ∈ H)⇒ ∀i; 1 ≤ i ≤ n; (commit$_{Ti}$ ∈H).

(abort$_T$ ∈ H) ⇒ ∀i; 1 ≤ i ≤ n; (abort$_{Ti}$ ∈ H).

Lemme 1 : Soit une transaction T_{st} du modèle « *AdapS* ». Si toutes les sous-transactions de T_{st} sont :

vitales, non remplaçables, non ré-exécutables,

et au plus une sous-transaction pivot, toutes les autres sont quasi-compensables.

Alors, T_{st} assure l'atomicité stricte.

Pour démontrer le lemme 1, nous démontrons d'abord le lemme1.1 sur la validation d'une transaction T_{st}, et après nous démontrons le lemme1.2 sur l'annulation d'une transaction T_{st}, ensuite nous déduisons que T_{st} assure l'atomicité stricte.

Lemme 1.1: La validation d'une transaction T_{st}.

Soit H l'historique d'une transaction T_{ST} avec n transactions composantes.

((commit$_{Tst}$ ∈ H) ⇒ ∀i; 1 ≤ i ≤ n (commit$_{Ti}$ ∈H))

La validation d'une transaction globale T_{st}, implique la validation de toutes les transactions composantes T_i associées.

Preuve du Lemme 1.1: Si T_{st} valide, l'ensemble de toutes ses transactions composantes doit valider.

Rappelons que les transactions composantes sont toutes vitales, au plus une sous transaction pivot, et toutes les autres sont quasi-compensables.

Etant donné que :

L'axiome (9) dit que la validation de la transaction parent dépend de :

la préparation de toutes les sous transactions quasi-compensables vitales,

la validation de la sous-transaction pivot vitale.

L'axiome (11) oblige les transactions quasi-compensables vitales à valider après la validation de la transaction parent.

L'axiome fondamental III précise qu'une transaction ne peut pas être terminée par deux événements différents.

D'après 1,2 et 3 nous pouvons conclure que la validation de la transaction T_{st} implique la validation de toutes les sous-transactions composantes.

Alors: $((commit_{Tst} \in H) \Rightarrow \forall i;\ 1 \leq i \leq n\ (commit_{Ti} \in H))$

Lemme 1.2: L'annulation d'une transaction T_{st} *avec atomicité stricte.*

Soit H l'historique d'une transaction T_{st} avec n transactions composantes.

$(abort_{Tst} \in H) \Rightarrow \forall i; 1 \leq i \leq n;\ ((\ abort_{Ti} \in H)$

L'annulation de T_{st} implique l'annulation de toutes ses transactions composantes associées.

Preuve du Lemme 1.2:

Nous étudions l'effet d'annulation d'une transaction globale T_{st} sur les transactions composantes selon leurs propriétés :

Cas 1. Les sous-transactions quasi-compensables :

L'axiome (8) oblige les sous-transactions quasi-compensables à ne pas valider avant leurs parents. Si le parent annule alors toutes les $T_i \in$ *QComp-Trs* en cours[10] seront annulées. Donc : $(abort_{Tst} \in H) \Rightarrow (abort_{Ti} \in H)$

Cas 2. La sous-transaction pivot vitale :

L'axiome (9) précise que la validation de la sous-transaction pivot vitale implique la validation de son parent.

Alors, ce n'est pas possible d'annuler la transaction parent après la validation du pivot.

D'après les deux cas (1 et 2) nous pouvons conclure que l'annulation d'une transaction parent implique l'annulation de toutes les sous-transactions qui sont en cours d'exécution: $(abort_{Tst} \in H) \Rightarrow \forall i; 1 \leq i \leq n;\ (\ abort_{Ti} \in H)$.

[10] L'axiome fondamental II spécifie que seules les transactions initiées sont annulées.

Preuve du Lemme 1 :

Rappelons que le lemme 1 annonce que T_{st} assure l'atomicité sémantique en satisfaisant les deux conditions de la définition 02.

1. La condition 1 de la définition 02 (si T_{st} valide, toutes les sous-transactions *Ti* doivent aussi valider) est satisfaite par le Lemme 1.1.

2. La condition 2 de la définition 02 (si T_{st} annule, toutes les sous-transactions *Ti* doivent annuler) est satisfaite par le Lemme 1.2.

Alors, nous pouvons déduire que la transaction T_{st} caractérisée par que toutes ses sous-transactions sont vitales, non remplaçables, non ré-exécutables, et au plus une sous-transaction pivot, toutes les autres sont quasi-compensables, assure l'atomicité stricte.

3.2.2. Atomicité sémantique d'une transaction *« AdapS »*

Afin de montrer que les transactions « *AdapS* » peuvent avoir la propriété d'atomicité sémantique, nous commençons par définir la propriété d'atomicité sémantique (définition 03). Puis, nous décrivons les conditions pour qu'une transaction T_{sm} du modèle « *AdapS* » assure une atomicité sémantique (Lemme 2). Ensuite, nous spécifions la validation de la transaction T_{sm} (Lemme 2.1) ainsi que son annulation (Lemme 2.2). Et enfin, nous déduisons l'atomicité sémantique de la transaction T_{sm}.

Définition 03 : Atomicité sémantique.

Une transaction T assure l'atomicité sémantique*:*

si T est validée, toutes les transactions composantes Ti doivent aussi être validées, et

si T est annulée, toutes les transactions Ti validées doivent soit être annulées soit être compensées.

Autrement dit:

$(commit_T \in H) \Rightarrow \forall i; 1 \leq i \leq n; (commit_{Ti} \in H)$

$(abort_T \in H) \Rightarrow \forall i;\ 1 \leq i \leq n; \forall j;\ 1 \leq j \leq n; i \neq j\ ((abort_{Ti} \in H) \wedge (commit_{Tj} \rightarrow commit_{comp\text{-}Tj}))$. *où comp-$T_j$ est la transaction de compensation de T_j.*

Lemme 2 : Soit une transaction T_{sm} du modèle « *AdapS* ». Si toutes les sous-transactions de T_{sm} sont :

vitales, non remplaçables, non ré-exécutables,

et au plus une sous-transaction pivot, toutes les autres sont quasi-compensables ou compensables.

Alors, T_{ST} assure l'atomicité sémantique.

Pour démontrer le lemme 2, nous démontrons d'abord le lemme2.1 sur la validation d'une transaction T_{sm}, et après nous démontrons le lemme2.2 sur l'annulation d'une transaction T_{sm}, ensuite nous déduisons que T_{sm} assure l'atomicité sémantique.

Lemme 2.1: La validation d'une transaction T_{sm}.

Soit H l'historique d'une transaction T_{sm} avec n transactions composantes.

$((commit_{Tsm} \in H) \Rightarrow \forall i;\ 1 \leq i \leq n\ (commit_{Ti} \in H))$

La validation d'une transaction globale T_{sm}, implique la validation de toutes les transactions composantes T_i associées.

Preuve du Lemme 2.1: Si T_{sm} valide, l'ensemble de toutes ses transactions composantes doit valider.

Rappelons que les transactions composantes sont toutes vitales, au plus une sous transaction pivot, et les autres sont soit compensables, soit quasi-compensables.

Etant donné que :

L'axiome (9) dit que la validation de la transaction parent dépend de :

la validation de toutes les sous-transactions compensables vitales,

la préparation de toutes les sous transactions quasi-compensables vitales,

la validation de la sous-transaction pivot vitale.

L'axiome (11) oblige les transactions quasi-compensables vitales à valider après la validation de la transaction parent.

L'axiome fondamental III précise qu'une transaction ne peut pas être terminée par deux événements différents.

D'après 1,2 et 3 nous pouvons conclure que la validation de la transaction T_{sm} implique la validation de toutes les sous-transactions composantes.

Alors: $((commit_{Tsm} \in H) \Rightarrow \forall i;\ 1 \leq i \leq n\ (commit_{Ti} \in H))$

Lemme 2.2: L'annulation d'une transaction T_{sm}.

Soit H l'historique d'une transaction T_{sm} avec n transactions composantes.

$(abort_{Tsm} \in H) \Rightarrow \forall i; 1 \leq i \leq n; \forall j; 1 \leq j \leq n; i \neq j;\ ((\ abort_{Ti} \in H) \wedge (commit_{Tj} \rightarrow commit_{comp\text{-}Tj}))$

L'annulation de T_{sm} implique l'annulation ou la compensation de toutes les transactions composantes associées.

Preuve du Lemme 2.2:

Nous étudions l'effet d'annulation d'une transaction globale T_{sm} sur ses transactions composantes selon leurs propriétés :

Cas 1. Les sous-transactions quasi-compensables :

L'axiome (8) oblige les sous-transactions quasi-compensables à ne pas valider avant leurs parents. Si le parent annule alors toutes les *Ti* ∈ *QComp-Trs* en cours seront annulées. Donc : $(abort_{Tsm} \in H) \Rightarrow (abort_{Ti} \in H)$

Cas 2. Les sous-transactions compensables :

L'axiome (7) garanti l'annulation des sous-transactions compensables qui n'ont pas encore validé en cas d'annulation du parent. $(abort_{Tsm} \in H) \Rightarrow (abort_{Ti} \in H)$

Alors que l'axiome (13) impose la validation des transactions compensatrices pour les sous-transactions compensables validées, si leurs parents annulent. Donc : $(abort_{Tsm} \in H) \Rightarrow (commit_{comp\text{-}Ti} \in H)$.

Cas 3. La sous-transaction pivot vitale :

L'axiome (9) précise que la validation de la sous-transaction pivot vitale implique la validation de son parent.

Alors, ce n'est pas possible d'annuler la transaction parent après la validation du pivot.

D'après les cas 1, 2 et 3, l'annulation d'une transaction T_{sm} implique l'annulation de toutes les sous-transactions qui sont en cours d'exécution qu'elles soient quasi-compensables ou compensable, et la compensation des sous-transactions compensables validées:

$$(abort_{Tsm} \in H) \Rightarrow \forall i; 1 \leq i \leq n; \forall j; 1 \leq j \leq n;\ i \neq j; ((abort_{Ti} \in H) \wedge (commit_{Tj} \rightarrow commit_{comp\text{-}Tj}))$$

Preuve du Lemme 2 :

Rappelons que le lemme 2 annonce que T_{sm} assure l'atomicité sémantique en satisfaisant les deux conditions de la définition 3.

1. La condition 1 de la définition 3 (si T_{sm} valide, toutes les sous-transactions *Ti* doivent aussi valider) est satisfaite par le Lemme 2.1.

2. La condition 2 de la définition 3 (si T_{sm} annule, toutes les sous-transactions *Ti* doivent soit annuler, soit compenser) est satisfaite par le Lemme 2.2.

Alors, nous pouvons déduire que la transaction T_{sm} caractérisée par que toutes ses sous-transactions sont vitales, non remplaçables, non ré-exécutables, et au plus une sous-transaction pivot, toutes les autres sont quasi-compensables ou compensable, assure l'atomicité sémantique.

3.2.3. Atomicité relâchée d'une transaction *« AdapS »*

Afin de montrer que les transactions « *AdapS* » peuvent avoir la propriété d'atomicité relâchée, nous commençons par définir l'atomicité relâchée (définition 04). Puis, nous décrivons les conditions pour qu'une transaction T_R du modèle « *AdapS* » assure une atomicité relâchée (Lemme 3). Ensuite, nous spécifions la validation de la transaction T_R (Lemme 3.1) ainsi que son annulation (Lemme 3.2). Enfin, nous obtenons l'atomicité relâchée de T_R.

Définition 04 : Atomicité relâchée.

Une transaction T assure l'atomicité relâchée:

si T est validée, toutes les transactions composantes Ti vitales doivent aussi être validées, et

si T est annulée, toutes les transactions Ti validées doivent être annulées.

Autrement dit:

$(commit_T \in H) \Rightarrow \forall i; 1 \leq i \leq n; (commit_{Ti}\ (r1, r2, r3, vitale) \in H)$

$(abort_T \in H) \Rightarrow \forall i;\ 1 \leq i \leq n; \forall j;\ 1 \leq j \leq n; i \neq j\ ((abort_{Ti} \in H) \wedge (commit_{Tj} \rightarrow commit_{comp\text{-}Tj}))$. *où comp-Ti est la transaction de compensation de Ti.*

Lemme 3 : Soit une transaction T_R du modèle « *AdapS* ». Si toutes les sous-transactions de T_R sont :

Vitales ou non, remplaçables ou non, ré-exécutables ou non,

et au plus une sous-transaction pivot, toutes les autres sont compensables ou quasi-compensables.

Alors, T_R assure l'atomicité relâchée.

Pour démontrer le lemme 3, nous démontrons d'abord le lemme3.1 sur la validation d'une transaction T_R, et après nous démontrons le lemme3.2 sur l'annulation d'une transaction T_R, ensuite nous déduisons que T_R assure l'atomicité stricte.

Lemme 3.1: La validation d'une transaction T_R

Soit H l'historique d'une transaction T_R avec n transactions composantes.

$((commit_{TR} \in H) \Rightarrow \forall i;\ 1 \leq i \leq n\ (commit_{Ti}\ (r1,r2,\ r3,\ vitale) \in H))$

La validation d'une transaction globale T_R, implique la validation de toutes les transactions composantes vitales T_i associées.

Preuve du Lemme 3.1: Si T_R valide, l'ensemble de ses sous-transactions vitales doit valider.

Rappelons que l'ensemble de sous-transactions vitales comporte l'ensemble de sous-transactions quasi-compensables vitales, l'ensemble de sous-transactions compensables vitales et la seule sous transaction pivot vitale.

Etant donné que :

L'axiome (9) dit que la validation de la transaction parent dépend de :

la validation de toutes les sous-transactions compensables vitales,

la préparation de toutes les sous transactions quasi-compensables vitales,

la validation de la sous-transaction pivot vitale.

L'axiome (11) oblige les transactions quasi-compensables vitales à valider après la validation de la transaction parent.

L'axiome fondamental III précise qu'une transaction ne peut pas être terminée par deux événements différents.

D'après 1,2 et 3 nous pouvons conclure que la validation de la transaction globale implique la validation de toutes les sous-transactions vitales composantes.

Alors: *$((commit_{TR} \in H) \Rightarrow \forall i;\ 1 \leq i \leq n\ (commit_{Ti}\ (r1,r2,\ r3,\ vitale) \in H))$*

Lemme 3.2: L'annulation d'une transaction T_R

Soit H l'historique d'une transaction T avec n transactions composantes.

$(abort_{TR} \in H) \Rightarrow \forall i; 1 \leq i \leq n;\ \forall j; 1 \leq j \leq n;\ i \neq j;\ ((\ abort_{Ti} \in H) \wedge (commit_{Tj} \rightarrow commit_{comp\text{-}Tj}))$

L'annulation de T_R implique l'annulation ou la compensation de toutes les transactions composantes associées (vitales ou non).

Preuve du Lemme 3.2:

Nous étudions l'effet d'annulation d'une transaction globale T sur ses transactions composantes selon leurs propriétés :

Cas 1. Les sous-transactions quasi-compensables :

L'axiome (8) oblige les sous-transactions quasi-compensables à ne pas valider avant leurs parents. Si le parent annule alors toutes les *Ti* ∈ *QComp-Trs* en cours seront annulées. Donc : $(abort_{TR} \in H) \Rightarrow (abort_{TR} \in H)$

Cas 2. Les sous-transactions compensables :

L'axiome (7) garanti l'annulation des sous-transactions compensables qui n'ont pas encore validé en cas d'annulation du parent. $(abort_{TR} \in H) \Rightarrow (abort_{Ti} \in H)$

Alors que l'axiome (13) impose la validation des transactions compensatrices pour les sous-transactions compensables validées, si leurs parents annulent. Donc : $(abort_{Tp} \in H) \Rightarrow (commit_{comp\text{-}Ti} \in H)$.

Cas 3. Les sous-transactions pivots :

La sous-transaction pivot vitale :

L'axiome (9) précise que la validation de la sous-transaction pivot vitale implique la validation de sa transaction parent.

Les sous-transactions pivots non vitales :

L'axiome (12) implique que la validation des sous-transactions pivots non-vitales se lance après la validation de leur parent.

D'après 1 et 2, nous pouvons conclure que ce n'est pas possible d'annuler la transaction parent après la validation d'une transaction pivot.

D'après les cas 1, 2 et 3, l'annulation d'une transaction T_R implique l'annulation de toutes les sous-transactions qui sont en cours d'exécution qu'elles soient quasi-compensables ou compensable, et la compensation des sous-transactions compensables validées:

$(abort_T \in H) \Rightarrow \forall i; 1 \leq i \leq n; \forall j; 1 \leq j \leq n; i \neq j; ((abort_{Ti} \in H) \wedge (commit_{Tj} \rightarrow commit_{comp\text{-}Tj}))$

Preuve du Lemme 3 :

Rappelons que le lemme 3 annonce que T_R assure l'atomicité relâchée en satisfaisant les deux conditions de la définition 4.

1. La condition 1 de la définition 4 (si T_R valide, toutes les sous-transactions *Ti vitales* doivent aussi valider) est satisfaite par le Lemme 3.1.

2. La condition 2 de la définition 4 (si T_R annule, toutes les sous-transactions T_i doivent soit annuler, soit compenser) est satisfaite par le Lemme 3.2.

Alors, nous pouvons déduire que vue que toutes ses sous-transactions sont vitales, non remplaçables, non ré-exécutables, et au plus une sous-transaction est pivot, toutes les autres sont quasi-compensables, la transaction T_R assure l'atomicité relâchée.

4. Conclusion

Dans cet article, nous nous sommes attelés à fournir un support pour les propriétés transactionnelles dans les services web. Pour cela, nous avons étudié les aspects transactionnels rencontrées dans ce domaine. Cette étude nous a permis de conclure sur le besoin d'un modèle flexible et adaptable pour la composition de services web avec propriétés transactionnelles. L'état de l'art réalisé concernant les différentes solutions en rapport avec cette thématique nous a permis de constater qu'elles ne répondent pas parfaitement aux besoins transactionnels envisagés. Ainsi, nous avons proposé notre approche à travers le modèle «*AdapS* » (*Adaptable Composite Web Services*) et sa formalisation. Nous avons également conçu un protocole de validation *apWST* (*Adaptable Protocol for Web Services Transactions*) et une architecture basée sur le concept de politique afin de supporter le modèle «*AdapS*» qui fera l'objet d'un autre article et nous travaillons actuellement sur la mise en œuvre d'un prototype.

Référence bibliographiques

[1] M.Attard, « Ubiquitous Web Services », proceedings of the first Computer Science Annual Workshop (CSAW'03), 2003.

[2] C.Ba , «Composition des services web avec PEWS : approche par la théorie des traces », Thèse de doctorat en informatique, Université François Rabelais Tours, 24 novembre 2008.

[3] S.Bhiri, « Approche Transactionnelle pour assurer des compositions fiables de services web » Thèse de doctorat en informatique, Université Henri Poincaré-Nancy 1, octobre 2005.

[4] Chrysanthis P.K., Ramamritham K., ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior, ACM SIGMOD, 1990.

[5] Chrysanthis P. K., Ramamritham K., Synthesis of extended transaction models using ACTA, ACM Transactions on Database Systems (TODS), vol. 19, Issue 3, September 1994, pp. 450 – 491.

[6] J.Daniel, « Services Web – Concepts, techniques et outils ». Editions Vuibert Informatique, Paris 2003. ISBN 2-7117-4813-8.

[7] H.Duarte, M-C.Fauvet, M.Dumas, B.Benatallah, « Vers un modèle de composition de services web avec propriétés transactionnelles », Revue Ingénierie des Systèmes d'Information - Numéro spécial Services Web, vol10, no3/2005, pp. 9-28, 2005.

[8] H.DUARTE-AMAYA, « Tcows : Canevas pour la composition de services web avec propriétés transactionnelles», Thèse de doctorat en informatique, Université JOSEPH FOURIER France, 13 Novembre 2007.

[9] J. El Haddad, O. Spanjaard, «Composition de services Web et équité vis-à-vis des utilisateurs finaux », 10ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, ROADEF 2009 Nancy, 10-12 février 2009

[10] T.HANH, «Coordination adaptative de services à base de contrats », Thèse de doctorat en informatique, Université JOSEPH FOURIER France, 2009. Institut Polytechnique De Grenoble.

[11] T.Jin & S.Goschnick (2003) Utilizing Web Services in an Agentbased Transaction Model (ABT). Proceedings, Workshop on Web Services and Agent-based Engineering, at the AAMAS-2003 conference, Melbourne, Australia.

[12] S.Krakowiak ,T.Coupaye, V.Quéma, L.Seinturier, J.-B. Stefani, M.Dumas, M.-C.Fauvet, P. Déchamboux, M. Riveill, A. Beugnard, D. Emsellem, D. Donsez, «Intergiciel et Construction d'Applications Réparties ». Chapitre 04 : Les services web, version du 12 juin 2008, disponible sur : <http://www2.lifl.fr/icar/main-onebib.pdf>.(dernière visite: Février 2010)

[13] N.Nouali-Taboudjemat , « Modèles et techniques de Transactions adaptable pour les environnements mobiles » Thèse de doctorat d'état en informatique, USTHB, Alger, N° 09/2007-E/IN, 17 Novembre 2007.

[14] OASIS Standard, «Web Services Coordination (WS-Coordination) Version 1.2 », 02 févier 2009. Disponible sur (Dernière visite : Mars 2011): http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os.pdf