Coordination sécurisée des services

Thi Huong Giang Vu

Groupe Grenoble INP - Grenoble Institute of Technology

No attribué par la bibliothèque

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INSTITUT POLYTECHNIQUE DE GRENOBLE

Spécialité : « Informatique »

préparée au **L**aboratoire d'**I**nformatique de **G**renoble (LIG)

dans le cadre de

l'École Doctorale « **M**athématiques, **S**ciences et **T**echnologies de l'**I**nformation, **I**nformatique »

présentée et soutenue publiquement

par

Thi Huong Giang VU

Le 18 novembre 2008

Titre :

Coordination sécurisée des services

(Secure coordination of services)

Directeur de thèse : Mme. Christine COLLET

JURY

Mme. Marie-Laure POTET,	Président
M. Djamal BENSLIMANE,	Rapporteur
M. Claude GODART,	Rapporteur
M. Hervé VERJUS,	Examinateur
Mme. Christine COLLET,	Directeur de thèse
Mme. Genoveva VARGAS-SOLAR,	Co-encadrant de thèse

Remerciements

C'est avec mon enthousiasme le plus vif et le plus sincère que je voudrais exprimer ma profonde gratitude à tous ceux qui m'ont aidée à mener à bien cette thèse.

Tout d'abord, je tiens à remercier ma directrice de thèse, Mme. Christine Collet, qui m'a accueillie dans l'équipe HADAS du laboratoire LIG. Je la remercie pour m'avoir aidée et encouragée tout au long de cette thèse. Je voudrais la remercier plus particulièrement pour tous les conseils importants et judicieux qu'elle m'apporte, ainsi que sa compréhension et ses soutiens (moral et financier).

Je remercie ma co-encadrante, Mme. Genoveva-Vargas Solar pour avoir suivi l'évolution de ma thèse avec intérêt. Je la remercie également pour ses remarques et ses corrections attentives.

Je tiens à remercier les personnes qui m'ont honoré en participant à mon jury : Mme. Marie-Laure Potet (Président et Examinateur), M. Djamal Benslimane et M. Claude Godart (Rapporteurs) et M. Hervé Verjus (Examinateur).

Je remercie le Centre National de la Recherche Scientifique et l'équipe HADAS pour m'avoir financée et m'avoir donnée des matériels essentiels pour réaliser cette thèse.

Je tiens à remercier tous les membres de l'équipe HADAS pour une très bonne ambiance de travail. J'adresse mes remerciements à Fabrice, Alexandre, Christophe, Sattisvar, Benjamin et Rémi qui m'ont beaucoup aidée à relire le manuscrit et à clarifier mes explications (écrites et orales). Plus particulièrement Christophe pour les discussions utiles sur mon travail de thèse.

Je remercie également mes collègues du LIG (Hanh, Diana, Victor, Seif, Carlos, Hien, Amal, Christine, Alexandra) pour l'affectueuse amitié dont ils ont toujours fait preuve, ainsi que tous les autres membres du LIG qui m'ont apportée leur aide de manière chaleureuse.

Je remercie aussi très vivement mes amis au Vietnam (vous êtes très nombreux et je ne peux pas tous vous citer), en France (Hai, Oanh, Cristina, Nguyen, Naga, Thanh, Laurent) ou ailleurs (Quan, Alberto, Victor, Angela, Pili, Lidia) qui m'ont beaucoup encouragée dans des moments difficiles de mon parcours de thésarde et pour leur soutien dépassant les frontières et les nationalités. Ils rendent ma vie moins dure.

La poursuite de cette thèse implique une longue période d'éloignement envers mes engagements pédagogiques à l'HUT ; je remercie donc vivement toutes les personnes là-bas pour leur assistance de l'obtention de permission pour la faire.

J'adresse mes remerciements à ma famille, qui me soutient toujours et qui a toujours une forte confiance en ma réussite. Aussi, je remercie de tout mon cœur mon mari (pour tout et pour rien).

Merci à toutes et à tous pour leur participation, leurs messages de soutien, à l'occasion de la soutenance, qui me touchent beaucoup.

A la mémoire de tous ceux que j'ai perdu.

Grenoble, 16 novembre 2008

Thi-Huong-Giang Vu

Résumé

Une application à base de services est construite à partir d'activités ; chaque activité correspond à un appel (par le biais d'une infrastructure de communication) à une fonction exportée par un service existant. Les activités sont coordonnées par un plan précisant les instructions à réaliser. Pour sécuriser de telles applications à base de services, les travaux actuels se focalisent sur la sécurité au niveau des services utilisés et au niveau de la communication entre ces services. Cependant, les mesures de sécurité à ces niveaux se gèrent mal au niveau du plan de coordination. Cette thèse concerne la coordination sécurisée pour la sécurité des applications à base de services. Elle considère un niveau de sûreté de fonctionnement pour (i) les activités d'un plan de coordination ; (ii) les données échangées entre services ; et (iii) les fonctions de services à appeler. La coordination sécurisée est définie à partir de trois concepts clés : (i) les activités à exécuter avec certaines propriétés dans le cadre de la coordination sécurisée, (ii) les contraintes associées aux activités régissant différents aspects considérés de la coordination sécurisée ; et (iii) les journaux de coordination construits à base de preuves d'exécution des activités. Un plan de coordination sécurisée d'activités est donc décrit sous forme d'un ensemble d'activités dont les contraintes devant être satisfaites sont définies par des formules logiques bien formées, correctement associées, cohérentes et évaluables. L'exécution d'un tel plan correspond à exécuter ses activités en évaluant les contraintes. Elle consiste également à évaluer si les appels aux fonctions de différents services se sont bien passés. L'exécution d'un plan peut s'adapter aux nouveaux besoins et aux changements (de propriétés, de contraintes, d'évaluateurs de contraintes, etc.) intervenant lors de l'exécution même du plan. Les contributions majeures de la thèse portent sur la définition d'un modèle pour la description de plan de coordination sécurisée d'activités, d'un modèle d'évaluation dynamique de ces plans et d'un canevas logiciel qui supporte la spécification, la transformation, l'exécution et la gestion des plans.

Mots-clés

Service, Application à base de services, Coordination sécurisée, Dynamicité

Abstract

Service-based application is built out of activities ; each activity corresponds to an invocation (through a communication infrastructure) of an existing service's function. Activities are coordinated by a plan specifying the instructions to be realized. To secure such applications, current works consider the security at the service level and at the service communication level. However, security measures at these levels are mapped inadequately at the coordination level. This thesis focuses on the secure coordination for securing service-based applications. We consider a functional safety level for (i) activities of a coordination plan ; (ii) data exchanged between services ; and (iii) functions of services to be invoked. Secure coordination is defined through three key concepts : (i) activities to be carried out with certain properties in the scope of coordination plan, (ii) constraints associated with the activities, which control various considered aspects of the plan ; and (iii) coordination logs, built from the execution proofs of activities. A secure coordination plan is described as a set of activities, whose constraints to be satisfied are well formed formulas, correctly associated, coherent and evaluable. Execute such a plan corresponds to execute activities by evaluating the constraints. It also consists in evaluating if the services' functions invocations are well done. The plan execution can adapt to the new requirements and changes (of properties, constraints, of constraint solvers, etc.) interfering at the execution time. This thesis contributes a description model of secure coordination plans ; a dynamic evaluation model of these plans and a framework which supports the specification, the transformation, the execution and the management of these plans.

Keywords

Service, Service-based application, Secure coordination, Dynamicity

Table des matières

Table des figures

Liste des Algorithmes

Liste des tableaux

Chapitre 1

Introduction

1.1 Motivation

Les progrès réalisés en matière de communication (explosion de l'Internet, architectures et protocoles normalisés d'interaction, réseaux hauts débits, etc.) facilitent de plus en plus l'accès à des services et le partage de gros volumes d'information multiforme. Ces services et informations concernent divers domaines (scientifiques, technologiques, commerciaux, etc.) et proviennent de différents fournisseurs (laboratoires, entreprises, compagnies, etc.). D'une part, une succession d'évolutions technologiques (web [jav, dot], bases de données [ora, jdb, MyS], etc.) a permis la construction de nouvelles applications qui exploitent ces informations. D'autre part, les fournisseurs contribuent à la construction de ces applications ; ils autorisent le contrôle des services afin de permettre leur exploitation. La notion de *service* offre ainsi un accès standardisé aux fonctions spécifiques de différents fournisseurs et un partage efficace des informations entre eux. Un service est identifié par un nom ; il exporte une interface et son mode d'emploi. L'interface définit un ensemble de signatures de fonctions du service. Le mode d'emploi définit les propriétés du service et les contraintes associées aux fonctions.

L'exploitation d'un service est effectuée par des appels à ses fonctions au travers de son interface. L'appel est conforme au mode d'emploi du service : (i) respect de la signature de la fonction, des propriétés requises pour l'exploitation du service et (ii) vérification de la satisfaction des contraintes internes du service. Une telle exploitation requiert de décrire pour chaque service :

- sa disponibilité : sa capacité à mener à terme ses fonctions, ses propriétés et ses contraintes ;
- son accessibilité : sa capacité à répondre à un appel d'une de ses fonctions en retenant ses propriétés et en respectant ses contraintes ;
- son inter-opérabilité : sa capacité à échanger des informations et à utiliser des informations qui ont été échangées (i) conformément à son mode d'emploi et (ii) de manière transparente par le biais d'une infrastructure de communication ;
- sa flexibilité : sa capacité à exécuter, à contrôler et à adapter ses fonctions, ses propriétés et ses contraintes.

L'exécution des appels requiert la présence d'une infrastructure de communication permettant l'interaction entre services (applicatifs). Nous pouvons considérer cette infrastructure comme une chaîne de services (d'interaction [FGM+06, PR85] et de réseau [tcp81, DH95, CDK+02, eth]) en charge de l'échange de messages entre les machines inter-connectées. En fait, les services applicatifs sont connectés à cette infrastructure de communication, via des services dits "d'interaction dédiés" [AP05, GM06]. Ces derniers confèrent certains critères autorisant l'exécution coordonnée entre services applicatifs, comme par exemple l'inter-opérabilité.

Les services d'interaction dédiés sont conçus suivant deux approches. En fonction de l'approche, nous parlons du partage d'une structure commune des informations (l'approche orientée données [FGM+06]) ou de l'échange de messages (l'approche orientée contrôle [iio04, soa06, rpc88]) entre les services applicatifs.

Une application à base de services est construite à partir d'activités ; chaque activité correspond à un appel à une fonction exportée par un service. Ces activités sont coordonnées selon un plan de coordination qui définit les contraintes d'ordonnancement des activités. L'exécution d'une telle application nécessite la *coordination* de services en tenant compte de la logique applicative et parfois d'autres aspects non-fonctionnels liés à la sécurité et à la sûreté de fonctionnement. La coordination de services nécessite qu'ils soient disponibles, accessibles, inter-opérables et éventuellement qu'ils soient flexibles. Les modèles de coordination existants se basent sur l'une des trois approches suivantes : orchestration [Pet81, wsb07, owl04, xpd05], chorégraphie [CG89, wsc05, wsc02] et collaboration [Fer97]. Un modèle de coordination permet de définir la logique applicative et la gestion de l'exécution coordonnée de services. Ceci se fait à travers la définition de trois aspects : ordonnancement, déclenchement et inter-dépendance de données. La façon de définition de ces aspects (explicite ou implicite) dépend de l'approche choisie. L'aspect ordonnancement exprime la relation temporelle de l'exécution de services. L'aspect déclenchement exprime la relation causale de l'exécution de services. L'aspect inter-dépendance de données représente la relation des données consommées/produites entre des services. Ces aspects sont généralement traités comme des propriétés et des contraintes sur le plan de coordination. En revanche, les aspects non-fonctionnels de l'exécution coordonnée de services, tels que la sécurité, sont souvent exclus du plan de coordination. Ces aspects sont considérés comme des propriétés ou des contraintes existantes au niveau de l'architecture qui exécute le plan de coordination [Cam04, RK07].

1.1.1 Problématique

Alors qu'une attention particulière a été donnée aux aspects fonctionnels (représentant la logique applicative) comme l'ordre d'exécution [wsb07, Pet81, CCF+] ou les données échangées [wsc05, CG89] entre les services, les aspects non-fonctionnels comme la sécurité, la sûreté de fonctionnement, etc. ont été peu considérés. Ils sont souvent traités différemment des autres aspects non-fonctionnels tels que les transactions, l'adaptation, etc. Les mesures répondant aux besoins applicatifs des aspects non-fonctionnels ne sont pas explicitement considérées lors de spécification des aspects fonctionnels de l'exécution coordonnée de services [AH96, HK03a, wss]. Par exemple, les mesures sécurisant le plan de coordination ne sont pas présentes dans la spécification du plan de coordination lui-même. Bien que les spécifications des aspects non-fonctionnels devraient être séparées des celles des aspects fonctionnels, l'hétérogénéité de ces spécifications ne nous permet pas d'appréhender d'une façon globale, cohérente et uniforme les aspects considérés afin de combiner leurs points forts dans un même contexte d'exécution.

L'exécution coordonnée de services implique en soi des vulnérabilités potentielles de sécurité qui sont introduites par différentes sources et qui sont exploitées par des services non autorisés. En raison de différences dans les politiques de contrôle et les implantations de services, cette exécution présuppose des niveaux de confiance appropriés à chaque service à coordonner. Par exemple, le service exécute seulement ceux qui sont publiquement déclarés. De même, cette exécution présuppose des niveaux de confiance sur les techniques supportées par l'infrastructure de communication. Par exemple, que les données échangées à travers des protocoles de com-

munication ne sont pas altérées. Les approches de sécurité traditionnelles ne s'appliquent pas en tant que telles pour l'exécution coordonnée, dans un contexte d'inter-opérabilité, d'inter-domaines d'applications et d'inter-gestion de services, où elles deviennent très coûteuses. Par conséquent, les applications à base de services sont plus sujettes à des menaces de sécurité que les systèmes d'information courants. Ces menaces sont dues aux vulnérabilités introduites par la spécification et l'implantation de ces applications et ce sur les trois plans suivants : service, communication et coordination. Dans ce contexte, la sécurité reflète des contraintes à satisfaire, sur la spécification et l'implantation de l'application à base de services. Cependant, les solutions ad-hoc actuelles ne consistent qu'en la sécurisation des composants spécifiques adressés à la mise en œuvre de l'exécution coordonnée. Cette sécurisation, quant à elle, se ramène à la mise en œuvre des solutions de sécurité vis-à-vis des services [wss, wst00] et de la communication [wss07, xml02, rfc02, ker02, sam05]. Elle ne peut pas être adaptée en fonction des besoins spécifiques des applications à base de services.

1.2 Objectifs de la thèse

L'objectif général de la thèse est de proposer des "outils" pour autoriser la définition et l'exécution de la **coordination sécurisée** des applications à base de services. Une coordination sécurisée permet de spécifier ce que fait une application à base de services. Elle permet également de prendre en compte à la fois les besoins des aspects fonctionnels et non-fonctionnels de l'application. Ces besoins sont considérés de manière homogène à partir de plusieurs points de vue : utilisateurs, fournisseurs, concepteurs, développeurs et techniciens.

Une coordination sécurisée permet d'assurer que les activités de l'application sont exécutées au bon moment, avec des paramètres corrects et par les services prévus. De plus, aucune activité non spécifiée ne sera exécutée ; toutes les activités spécifiées sont exécutées ; et l'exécution des activités est terminée comme prévue. Une coordination sécurisée a pour but de répondre aux besoins de sécurité suivantes :

1. **Authentification :**
   - L'identité de n'importe quelle activité est reconnue.
   - Les activités correspondent à des fonctions de services prévus pour être utilisés.
   - L'identité de n'importe quel service utilisé ou la source de n'importe quelle information utilisée dans le cadre du plan de coordination est reconnue.
2. **Autorisation :**
   - Les informations sensibles définissant l'exécution coordonnée de services ne sont pas révélées aux services non autorisés.
   - L'exécution d'une activité ne peut pas être démarrée ou observée par un service ou une ressource non autorisée.
   - Selon son authentification et son autorisation, un accès (aux informations ou aux fonctions de services) n'est pas illicite.
3. **Intégrité :** les informations de l'exécution coordonnée (qu'elles soient utilisées, stockées ou échangées), les activités à exécuter et les services utilisés ne sont pas modifiés, altérés ou détruits de manière imprévue ou accidentelle.
4. **Confidentialité :**
   - Toute information sensible n'est rendue compréhensible que par ce qui est prévu.

- La propagation des informations sensibles propres aux services ou définissant l'exécution coordonnée est contrôlée (quand, comment et à quel degré de confidentialité la propagation est autorisée).

5. **Non-répudiation :** il faut disposer de preuves pour vérifier a posteriori la validité de l'exécution d'une application à base de services, en particulier que :
   - les activités n'ont pas enfreint le plan (*trace*) ;
   - les activités ayant subi une attaque, les informations ou les services étant compromis par une menace effective sont ultérieurement détectés (*audit*) ;
   - un service a effectué une activité de manière certaine (*imputation*).

Afin de sécuriser les applications à base de services, notre travail se focalise sur la sécurité autour du plan de coordination. Nous supposons donc que la sécurité est assurée à l'égard des services utilisés et à l'égard de l'infrastructure de communication (chaîne de services d'interaction et de réseaux utilisés). Plus précisément, notre approche est construite en se basant sur les hypothèses suivantes :

- Les services utilisés exportent leurs interfaces et leurs modes d'emploi. Il n'y a pas de fonctions inconnues des exploiteurs légitimes, qui donnent un accès secret aux services (par exemple porte dérobée).
- Les services possèdent des propriétés et des contraintes internes de sécurité différentes. Par exemple, certains services utilisés supportent l'authentification de leurs utilisateurs, d'autres services ne la supportent pas.
- Les services mettent en application leurs propriétés et leurs contraintes de sécurité par des mesures hétérogènes (politique, modèle, architecture, mécanisme), par exemple différents mécanismes d'authentification.
- La confidentialité des données échangées lors de l'exécution coordonnée de services est assurée par les services sécurité dédiés ou génériques ; ces derniers sont associés à certains services de la chaîne de services d'interaction et de réseau de l'infrastructure de communication, par exemple les protocoles chiffrés.
- Les informations sensibles des applications à base de services sont persistantes durant l'exécution coordonnée de services et sont révélées d'une manière sécurisée grâce aux services de persistance.

Dans notre approche, sécuriser une application à base de services consiste à assurer la coordination sécurisée de ses activités. Nous proposons une coordination sécurisée à deux niveaux : spécification et exécution.

1.3 Contributions de la thèse

Considérer la coordination sécurisée de services à deux niveaux consiste :

Au niveau de la spécification : à contrôler les activités d'une application à base de services en fonction d'un plan de coordination sécurisée. Du point de vue du plan de coordination sécurisée, une activité :

- appartient à un plan de coordination sécurisée ;
- possède des propriétés nécessaires pour l'exécution de ce plan ;
- a conscience des services servant à son exécution dans ce plan ;

- est conditionnée par des contraintes de ce plan qui doivent être satisfaites avant et/ou après l'appel ;
- fournit des preuves d'exécution qui sont archivées dans des journaux de coordination ; ces preuves servent à l'exécution des autres activités du plan ainsi qu'aux analyses ultérieures.

Un plan de coordination sécurisée précise les instructions à réaliser sur les activités concernant :

- les aspects fonctionnels de coordination : l'ordonnancement, le déclenchement et l'interdépendance de données, définis par des contraintes de coordination ;
- les aspects non-fonctionnels de sécurité : l'authentification, l'autorisation, l'intégrité et la non-répudiation, définis par des contraintes de sécurité.

Au niveau de l'exécution : à évaluer les contraintes (de coordination, de sécurité, etc.) du plan, et également à évaluer si l'exécution des appels aux fonctions des différents services s'est bien passée. L'exécution d'un plan de coordination sécurisée d'activités repose donc sur des évaluateurs de contraintes.

La sûreté de fonctionnement de cette exécution est assurée à la fois par (i) la satisfaction de contraintes (notamment de sécurité), et (ii) le traitement d'exception lors de l'occurrence d'une contrainte non satisfaisante.

Par ailleurs, l'exécution du plan s'adapte aux nouveaux besoins et changements (de propriétés, de contraintes, d'évaluateurs, etc.) qui peuvent apparaître durant cette exécution. Une telle adaptation s'exprime en terme les propriétés et les contraintes spécifiques du plan. Ces propriétés et contraintes, quant à elles, sont définies dans le plan, ou générées à partir du plan ou bien encore redéfinies lors de l'exécution du plan.

1.3.1 Le modèle MEO

Il s'agit d'un modèle de coordination sécurisée à base de contraintes [Vu06a, Vu06b, VCVS06a]. La spécification, l'exécution, l'adaptation et la sûreté de fonctionnement du plan de coordination sécurisée sont traitées autour des contraintes.

Le modèle permet de définir de manière récapitulative et indépendante les aspects fonctionnels et non-fonctionnels de l'exécution coordonnée de services. Ce modèle permet également d'homogénéiser ces aspects en vue du processus d'exécution interne d'une application à base de services (plan de coordination sécurisée) et en vue de son abstraction (un service avec nom, interface et mode d'emploi).

La coordination sécurisée est modélisée à partir des concepts suivants :

- **activité**s construisant l'application à base de services, à exécuter avec certaines **propriété**s dans le cadre du plan de coordination sécurisée ;
- **contrainte**s régissant les différents aspects de l'exécution du plan, à satisfaire avant ou après l'exécution d'appel dans le cadre des activités du plan ;
- **journal de coordination** archivant les **preuve**s **d'exécution** du plan.

Le langage associé à ce modèle permet de décrire le plan de coordination sécurisée en termes des informations qui s'apparentent aux types prédéfinis et qui correspondent aux valeurs reconnues. Par ailleurs, il permet de décrire les aspects (propriétés et/ou contraintes) devant être associés aux services utilisés. Il permet également de décrire les informations utiles à l'exécution du plan.

L'exécution d'un plan de coordination sécurisée d'activités consiste à évaluer ses contraintes par les algorithmes d'évaluation associés à ce modèle. Les différents aspects considérés de l'exécution coordonnée de services sont traités de manière unique (qui se base sur la satisfaction des contraintes correspondantes). La sûreté de fonctionnement de l'exécution est assurée à la fois par la satisfaction des contraintes du plan et le traitement dans le cas de non-satisfaction de contrainte.

1.3.2 Le canevas MEOBI pour la construction d'applications sécurisées à base de services

Nous proposons un canevas logiciel associé au modèle MEO appelé MEOBI. Le terme canevas désigne ici un ensemble d'interfaces avec une implantation partielle et leurs modes d'emploi. MEOBI est destiné aux différents intervenants à la construction d'une application sécurisée à base de services (voir la figure 1.1). Il permet de :

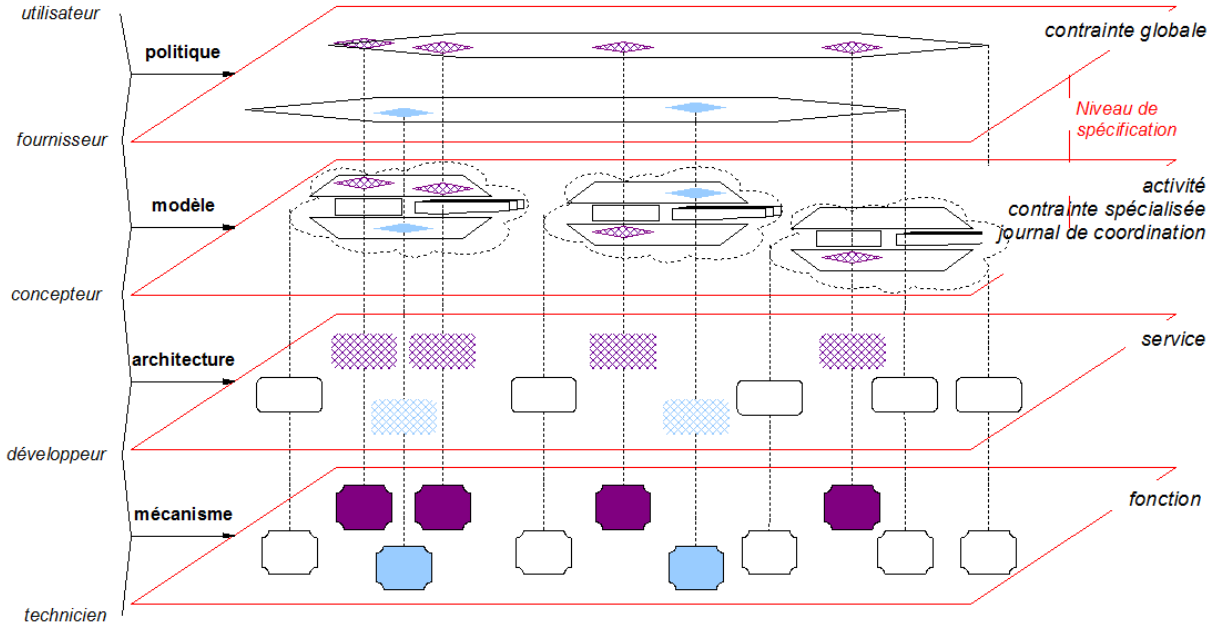


FIG. 1.1 – Coordination sécurisée

- Définir (en terme des contraintes globales de différents aspects) des politiques de contrôle à partir du point de vue de différents intervenants : utilisateurs, fournisseurs, architectes d'application, développeurs, techniciens.
- Définir et manipuler les informations définissant un plan de coordination sécurisée.
- Contrôler l'exécution du plan conformément aux contraintes globales pré-définies.
- Supporter l'instanciation des mécanismes de coordination, de sécurité sous forme des fonctions des services constitutifs de l'architecture d'exécution.
- Adapter les contraintes globales conformément aux propriétés et contraintes des services à coordonner, de leurs environnements d'exécution et de leur contexte de déploiement.
- Faciliter la transformation de contraintes globales en contraintes spécialisées concrètes.
- Guider les mécanismes pouvant être utilisés à différents niveaux dans la programmation, la coordination, la sécurité de services, leur exécution.

MEOBI fournit une architecture générale pour gérer et contrôler la coordination sécurisée [VCVS06b, VCCVS06]. Cette architecture décrit l'organisation des composants chargés de l'exécution co-

ordonnée de services utilisés ainsi que les interactions entre eux. Ces composants comprennent des contrôleurs et des évaluateurs de contrainte. Ils peuvent être combinés ensemble de manière dynamique. Un composant peut être spécialisé pour :

- Évaluer les contraintes spécifiées.
- Exécuter les activités spécifiées.
- Mettre en service le plan de coordination sécurisée.
- Implanter des politiques (coordination, sécurité, sûreté de fonctionnement).
- Ajouter les propriétés non-fonctionnelles souhaitées aux services.
- Faire communiquer l'application à base de service et les services utilisés.
- Surveiller l'exécution coordonnée.

Cette architecture s'adapte aux plans de coordination sécurisée d'activités définis par le modèle MEO. Elle exécute, contrôle et gère un plan en assurant son adaptation et sa sûreté de fonctionnement.

1.4 Organisation du document

La suite de ce document est organisée de la manière suivante. Le chapitre 2 présente l'état de l'art de la sécurité dans les applications à base de services. Il représente des solutions existantes sur différents aspects de sécurité (authentification, autorisation, intégrité, confidentialité, non-répudiation) qui servent à trois plans : coordination, communication et services. Le chapitre 3 présente MEO, notre modèle de coordination sécurisée. Il décrit les concepts nécessaires à la définition de la coordination sécurisée. Il décrit également les propriétés et les prédicats de base utiles pour définir les contraintes globales régissant les différents aspects de l'exécution du plan. Le chapitre 4 décrit le modèle d'exécution du plan de coordination sécurisée d'activités. Le chapitre 5 présente notre canevas MEOBI pour la construction des applications à base de services. Il montre aussi comment instancier ce canevas pour implanter un moteur d'exécution du plan. Finalement, le chapitre 6 conclut ce document en mettant l'accent sur les apports de notre travail et en donnant quelques perspectives pour des recherches futurs.

Chapitre 2

Sécurité dans les applications à base de services

Ce chapitre présente un état de l'art sur la sécurité dans les applications à base de services. De manière générale, sécuriser une application à base de services consiste à assurer la coordination sécurisée de ses activités. Cela signifie que la spécification et l'implantation de cette coordination n'introduit pas de vulnérabilités. Cela implique la spécification et l'implantation de mesures de sécurité : authentification, autorisation, intégrité, confidentialité et non-répudiation. La coordination sécurisée considère un niveau de sûreté de fonctionnement pour (i) des fonctions de services à appeler ; (ii) des données échangées entre services ; et (iii) des activités d'un plan de coordination.
Ce chapitre est organisé de la manière suivante. La section 2.1 décrit la façon dont la sécurité est redéfinie dans le contexte des applications à base de services. Les sections 2.2 à 2.6 présentent en détail les travaux actuels sur les aspects sécurité d'une coordination. Finalement, la section 2.7 montre jusqu'à quel point les travaux existants répondent aux besoins d'une coordination sécurisée, et analyse également leurs points faibles.

2.1 Application à base de services sécurisée

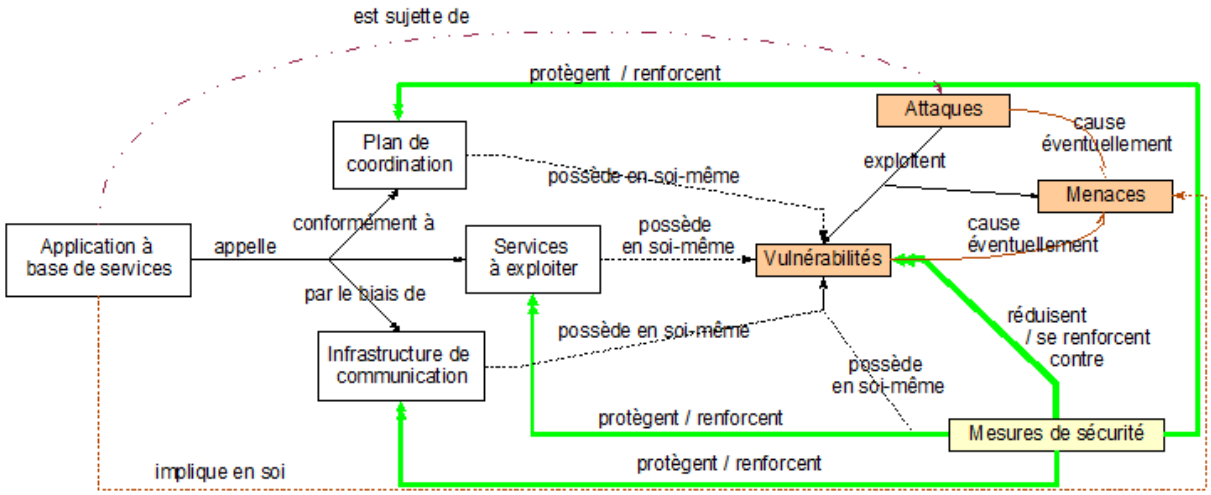


Fig. 2.1 – Application à base de services sécurisée

Une application à base de services est construite à partir d'activités ; chaque activité corres-

pond à l'appel à une fonction exportée par un service existant. Les ovales et les flèches de style fin et continu qui se trouvent à gauche de la figure 2.1 illustrent cette construction : une application appelle des services par le biais d'une infrastructure de communication et conformément à un plan de coordination. Le plan de coordination définit les contraintes d'ordonnancement des activités.

Une application à base de services est dite "sécurisée" si elle se renforce ou est protégée contre des menaces, des attaques et des vulnérabilités de manière contrôlable. Dans la figure 2.1, la relation entre ces dernières (illustrées par les carrés en haut et à droite de la figure) et les éléments construisant une application à base de services (les ovales à gauche de la figure) est illustrée par les flèches de style pointillé ou tiret. Les flèches de style gras et continu illustrent la protection et le renforcement des éléments construisant une application à base de services (ovales) par des mesures de sécurité (carré en bas à droite).

2.1.1 Menaces, attaques et vulnérabilités

Menace Il s'agit d'une situation qui peut causer une violation (intentionnelle ou accidentelle) de sécurité. Une application à base de services est menacée durant l'exécution de ses activités dans quatre situations possibles :

Interruption : les services ou les informations (stockées, utilisées ou échangées) deviennent accidentellement indisponibles ou inaccessibles pour l'exécution des activités.

Interception : les informations (stockées, utilisées ou échangées) sont révélées à ceux qui ne sont pas autorisés ; ceux qui n'ont pas l'autorité requise accèdent aux services.

Modification : les services sont changés d'une manière imprévue par rapport à la définition originale ; les informations (stockées, utilisées ou échangées) entre les services sont modifiées sans préavis ou de manière non autorisée.

Fabrication : les services sont ajoutés ou remplacés par d'autres services non souhaités ; les informations (stockées, utilisées ou échangées) sont générées sans préavis ou par des services non autorisés.

Attaque Généralement, une attaque peut être caractérisée selon son but (attaque passive/active), son commencement (attaque interne/externe) ou son exploitation (attaque directe/indirecte/hors ligne). Une application à base de services est sujette à plusieurs méthodes d'attaque résultant de sa modification ou sa révélation inattendue :

Denial of service : l'accès aux fonctions qu'un service fournit ou à ses informations stockées est empêché auprès de ceux qui sont autorisés.

Eavesdropping : les informations échangées par des services prévus dans le cadre de l'exécution coordonnée sont interceptées et révélées.

Gathering : les informations sensibles décrivant des services sont révélées ; les activités exécutées sont révélées.

Masquerading : l'identité d'un service est usurpée pour exécuter des activités malveillantes ou hostiles.

Infiltration : les activités sont exécutées et les informations sont échangées de manière illégale au nom d'un service prévu par un autre service prévu.

Tampering : les informations échangées par des services prévus dans le cadre de l'exécution coordonnée sont interceptées et modifiées.

Replaying : les informations interceptées sont redirigées afin de gagner les droits de l'expéditeur réel.

Vulnérabilité Cette notion désigne les ressources qui pourraient être exploitées pour violer la sécurité. Une application à base de services possède les types de vulnérabilité suivants :

Concernant l'aspect ordonnancement : des activités d'un plan de coordination ne sont pas exécutées dans le bon ordre.

Concernant l'aspect déclenchement : des activités ne sont pas exécutées par les services prévus ou ne sont pas exécutées au bon moment.

Concernant l'aspect interdépendance de données : des activités sont exécutées avec des paramètres incorrects.

Les ressources vulnérables sont souvent celles qui rendent réalisable l'exécution coordonnée de services et/ou qui répondent aux besoins applicatifs. Dans le cadre de l'application à base de services, ces ressources interviennent à trois niveaux (voir la figure 2.2) :

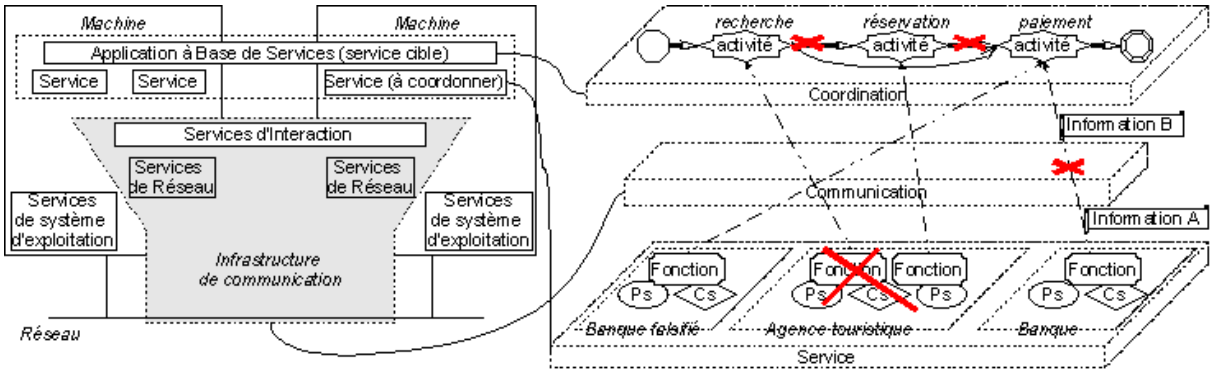


FIG. 2.2 – Vulnérabilités d'une application à base de services

Niveau des services : les ressources permettant la disponibilité, l'accessibilité et la flexibilité des services à exploiter sont vulnérables (par exemple les fonctions de contrôle d'un service à exploiter) ;

Niveau de la communication :

- les ressources permettant la communication orientée données ou orientée contrôle entre services sont vulnérables (par exemple les services d'interaction dédiés de chaque modèle de coordination).
- la chaîne de services d'interaction et de services de réseau d'une infrastructure de communication permettant l'inter-opérabilité entre services est vulnérable (par exemple les services dont les fonctions servent à l'exécution des services d'interaction dédiés de chaque modèle de coordination).

Niveau du plan de coordination : l'expression sémantique[1] et la gestion de l'exécution coordonnée de services conformément à l'une des trois approches de coordination : orchestration, chorégraphie ou collaboration (par exemple le moteur de coordination).

[1]Dorénavant, sauf mention explicite contraire, le terme "expression sémantique de l'exécution coordonnée de services" est utilisés d'une manière interchangeable avec le terme "logique applicative de l'application à base de services".

Elles influencent la sûreté de fonctionnement d'une application durant l'exécution coordonnée de ses services : il y a des activités correctes qui ne sont pas exécutées, il y a des activités incorrectes qui sont exécutées, ou il y a des activités dont l'exécution ne s'est pas terminée comme prévue.

En pratique, les menaces identifiées ci-dessus sont inévitables. Les attaques sont variées et elles exploitent des vulnérabilités concrètes. Leur succès dépend de la vulnérabilité qu'elles exploitent ainsi que de l'efficacité des mesures de sécurité appliquées.

2.1.2 Mesures de sécurité

La protection d'une application à base de services se ramène soit à la suppression de ses vulnérabilités, soit à la réduction de ses vulnérabilités et au renforcement de l'application elle-même. L'idée est d'assurer un certain degré de sûreté de fonctionnement pour :

- les fonctions de services à appeler dans le cadre des activités. Ceci est assuré par des mécanismes de sécurité supportés par les services appelés ;
- les données échangées entre services dans le cadre des activités. Ceci est assuré par des mécanismes de sécurité supportés par les services appelés, ainsi que par l'infrastructure de communication ;
- les activités d'un plan de coordination. Ceci est le reflet des mécanismes de sécurité supportés par les services appelés et par l'infrastructure de communication.

Les travaux actuels se focalisent sur les mesures de sécurité : politiques[2], modèles[3], architectures[4] et mécanismes[5]. Ces mesures sont toutes orientées sur une architecture d'exécution du plan de coordination. Plus précisément, le plan de coordination est supposé pré-établi conformément à un modèle de coordination (orchestration, chorégraphie, collaboration). Ce plan est exécuté par une *configuration* spécifique des éléments constitutifs de l'architecture d'exécution fournie par ce modèle de coordination. L'exécution du plan est conforme aux propriétés et contraintes prédéfinies par ce modèle. Les aspects sécurité sont traités autour des éléments constitutifs de cette architecture. Assurer la coordination sécurisée des activités d'une application à base de services ne consiste donc qu'à valider des propriétés et/ou des contraintes de sécurité sur une configuration spécifique de cette architecture. Autrement dit, il s'agit de sécuriser le *moteur d'exécution du plan de coordination*. Trois approches suivantes ont été proposées ; les travaux de chaque approche seront détaillés dans la suite de ce chapitre :

1. **Versions sécurisées** : cette approche, dite **centralisée** (voir la figure 2.3a), consiste à la mise en œuvre d'un ensemble ad-hoc de mesures sur tous les aspects de sécurité considérés. Ces mesures sont associées à la configuration exécutant le plan elle-même. Cette mise en œuvre est spécifique à un plan de coordination et elle est codifiée dans des éléments constitutifs de cette configuration. Les propriétés et contraintes de sécurité de tout aspect considéré ne seront validées que sur cette configuration.
2. **Configurations sécurisées** : cette approche, dite **décentralisée** (voir la figure 2.3b), consiste en la mise en œuvre d'un ensemble ad-hoc de mesures sur tous les aspects de

[2] "Security policy : A set of policy rules (or principles) that direct how a system (or an organization) provides security services to protect sensitive and critical system resources." [rfca]

[3] "Security model : A schematic description of a set of entities and relationships by which a specified set of security services are provided by or within a system." [rfca]

[4] "Security architecture : A plan and set of principles that describe (a) the security services that a system is required to provide to meet the needs of its users, (b) the system components required to implement the services, and (c) the performance levels required in the components to deal with the threat environment." [rfca]

[5] "Security mechanism : A method or process (or a device incorporating it) that can be used in a system to implement a security service that is provided by or within the system." [rfca]

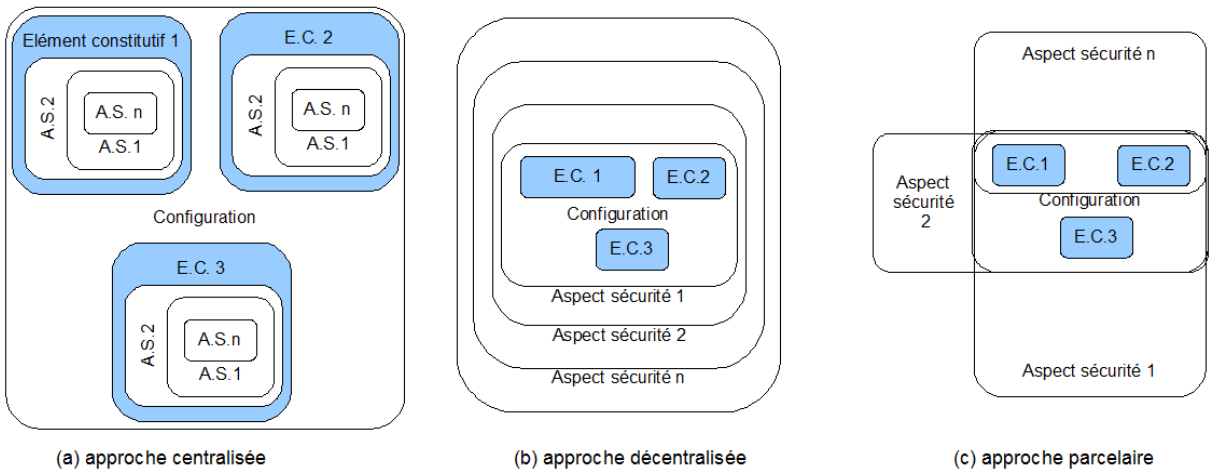


FIG. 2.3 – Approches de sécuriser l'application à base de services

sécurité considérés. Ces mesures sont associées à l'architecture d'un modèle de coordination concret. Elles servent à toutes les configurations possibles de cette architecture, dont le moteur d'exécution du plan. Une telle mise en œuvre assure que les propriétés des aspects sécurité considérés sont retenues et/ou les contraintes des aspects sécurité considérés sont satisfaites.

3. **Services dédiés/génériques de sécurité** : cette approche, dite **parcelaire** (voir la figure 2.3c), consiste en la mise en œuvre ad-hoc et indépendante des mesures de sécurité sur une partie seulement des aspects sécurité considérés. Ces mesures sont destinées aux éléments constitutifs d'une architecture d'un modèle de coordination concret. Les éléments qui implantent ces mesures font partie de la configuration exécutant un plan (comme des fonctions fournies par le moteur de coordination). Le cas échéant, ces éléments sont à l'extérieur de la configuration. Par exemple, des fonctions externes à appeler par le moteur de coordination ; ces fonctions sont fournies par des services qui sont soit spécifiquement conçus pour un aspect sécurité, soit génériques pour différents aspects sécurité.

Une application à base de services est finalement elle-même un service, qui doit être construit de manière réutilisable. Afin de ne pas réduire la disponibilité, l'accessibilité et la flexibilité au cours de la protection de l'application, les mécanismes de sécurité supportés par l'application et par les services qui la construisent doivent, eux aussi, être inter-opérables. Les mécanismes supportés par une application (selon ces trois approches) doivent donc être cohérents. Ils se ramènent à ceux qui sont supportés par et/ou appliqués sur les services utilisés (les services à coordonner, les services d'interaction et de réseau de l'infrastructure de communication). De plus, de tels mécanismes sont réalisables seulement dans des contextes de développement et de déploiement concrets qui possèdent eux-mêmes des mécanismes permettant d'assurer un certain niveau de sécurité.

2.1.3 Objectifs de la coordination sécurisée

Les aspects authentification, autorisation, intégrité, confidentialité et non-répudiation de la sécurité contribuent à la réduction des vulnérabilités et au renforcement d'une application à base de services. Les objectifs pour la coordination sécurisée sont définis en fonction des besoins sur ces cinq aspects sécurité.

1. **Authentification :**
   - L'identité de n'importe quelle activité est reconnue.
   - Les activités correspondent à des fonctions de services prévues pour être utilisées.
   - L'identité de n'importe quel service utilisé ou la source de n'importe quelle information utilisée dans le cadre du plan de coordination est reconnue.
2. **Autorisation :**
   - Les informations sensibles définissant l'exécution coordonnée des services ne sont pas révélées aux services non autorisés.
   - L'exécution d'une activité ne peut pas être démarrée ou observée par un service ou une ressource non autorisée.
   - Selon son authentification et son autorisation, un accès (aux informations ou aux fonctions de services) n'est pas illicite.
3. **Intégrité :** les informations de l'exécution coordonnée (qu'elles soient utilisées, stockées ou échangées), les activités à exécuter et les services utilisés ne sont pas modifiés, altérés ou détruits de manière imprévue ou accidentelle.
4. **Confidentialité :**
   - Toute information sensible n'est rendue compréhensible que par ce qui est prévu.
   - La propagation des informations sensibles propres aux services ou définissant l'exécution coordonnée est contrôlée (quand, comment et à quel degré de confidentialité la propagation est autorisée).
5. **Non-répudiation :** il faut disposer de preuves pour vérifier a posteriori la validité de l'exécution d'une application à base de services, en particulier que :
   - les activités n'ont pas enfreint le plan (*trace*) ;
   - les activités ayant subi une attaque, les informations ou les services étant compromis par une menace effective sont ultérieurement détectés (*audit*) ;
   - un service a effectué une activité de manière certaine (*imputation*).

2.2 Authentification[6]

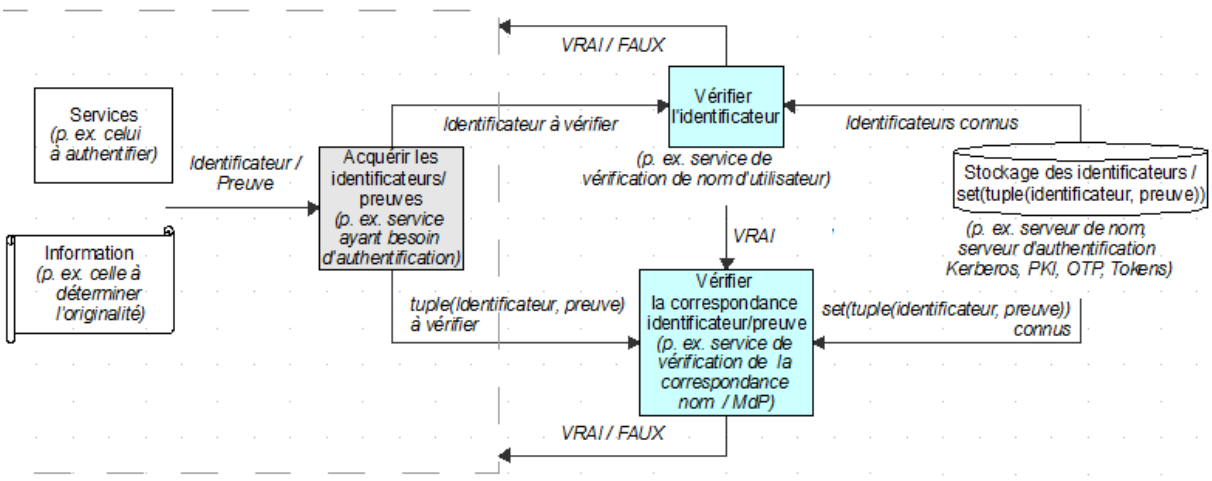


FIG. 2.4 – Principe d'authentification

[6] "The process of verifying a claim that a system entity or system resource has a certain attribute value." [rfca]

L'authentification consiste à vérifier la correspondance légitime entre l'identificateur (d'un service ou d'une ressource à authentifier) et une (ou plusieurs) preuve(s) démontrant la véracité de cet identificateur (voir la figure 2.4).

Les preuves à présenter sont secrètes ou sont visibles du point de vue de cet identificateur. Il peut s'agir d'une ou plusieurs des preuves suivantes :

- preuves abstraites que l'identificateur est le seul à connaître (par exemple un mot de passe, une réponse de challenge dans le cas où le mot de passe est oublié, un certificat PKI [rfc99a], etc.) ;
- preuves physiques uniques qui sont détenues par cet identificateur (par exemple un badge RSA [Riv02], une carte à puce, etc.) ; ou
- preuves biométriques ou physiques qui n'appartiennent qu'à cet identificateur (par exemple une empreinte, une image rétinienne, un mouvement, etc.) ;
- preuves permettant de déterminer la relation de confiance entre l'identificateur à authentifier et un certain identificateur authentifié (par exemple une proximité de connexion d'un équipement GPS).

Ces preuves sont vérifiées en les comparant avec des informations qui sont préalablement reconnues ou enregistrées.

La vérification dépend de la façon dont les identificateurs et les preuves sont construits. Elle dépend aussi de la façon dont le service ou la ressource à authentifier présente son identificateur et sa (ses) preuve(s), ainsi que du contexte de cette présentation. Des identificateurs, des preuves, ou éventuellement des informations à propos de leur construction ou de leur présentation, etc., sont tous considérés comme des propriétés d'authentification de services ou de ressources à authentifier. En pratique, plusieurs mécanismes d'authentification ont été proposés. Ces mécanismes ont trait aux algorithmes de comparaison, par exemple des mécanismes de chiffrement / déchiffrement, de signature, de certification. Ils ont trait également aux schémas d'échange des propriétés d'authentification, par exemple des protocoles d'échange - à sens unique, à deux sens, à trois sens - des certificats. Il s'agit de mécanismes prouvant que (i) l'identité d'un service ou d'une ressource est véritable (*peer entity authentication*) ou que (ii) la source d'une ressource ou d'une information est valide (*data origin authentication*).

Chiffrement Les mécanismes de chiffrement des informations échangées permettent de vérifier (i) l'origine des informations échangées en tant que message envoyé / reçu et (ii) l'authenticité des services en tant qu'expéditeur et destinataire de ces informations.

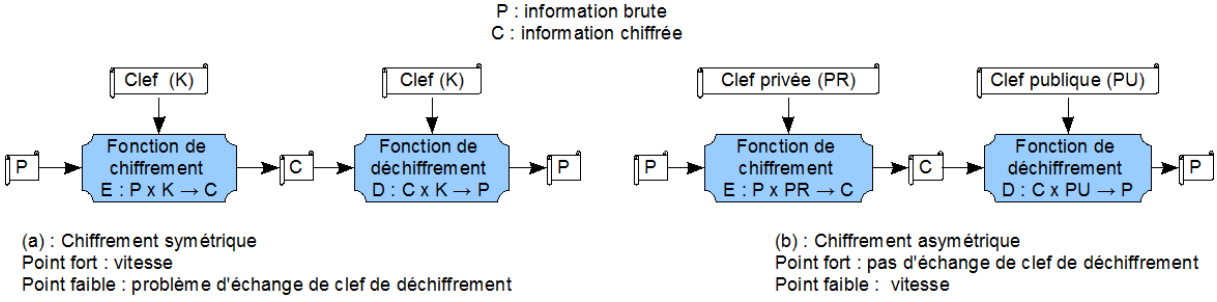


FIG. 2.5 – Chiffrement symétrique et asymétrique pour l'authentification

En utilisant une fonction de chiffrement symétrique telle que AES (Advanced Encryption Standard [fip01]) ou RC6 (Ron's Code #6 [Ra00]), un service qui détient une clef de chiffrement

peut à la fois chiffrer et déchiffrer des informations échangées et il doit cacher cette clef (voir la figure 2.5(a)) [7] .
En utilisant une fonction de chiffrement asymétrique qui se base sur l'usage d'une couple de clefs publique et privée (telle que RSA (Rivest Shamir Adleman [Riv02]) ou Diffie-Hellman [DH76]), les informations échangées sont chiffrées par une clef et déchiffrées par l'autre clef (voir la figure 2.5(b)). La clef publique est supposée connue par plusieurs services, et la clef privée est supposée connue que par un seul service. La clef de chiffrement (publique ou privée) est la preuve de l'expéditeur et la clef de déchiffrement (resp. privée ou publique) est la preuve du destinataire. S'il s'agit d'une clef privée, l'identificateur de l'expéditeur ou du destinataire qui la détient est reconnu [7] .

Signature Une *signature* est nécessaire pour authentifier ce qui chiffre et déchiffre des informations échangées. Une signature est créée par un service en chiffrant l'information avec sa clef privée, ou bien en chiffrant une empreinte de l'information avec cette clef. Tandis que la clef privée d'un service est utilisée pour créer la signature, la clef publique correspondante est utilisée pour vérifier la signature. La signature créée de cette façon est unique à une information et à la clef privée utilisée pour chiffrer cette information et elle ne peut pas être forgée. Le service qui détient la clef privée est reconnu en tant que signataire par le service qui détient la clef publique correspondante.

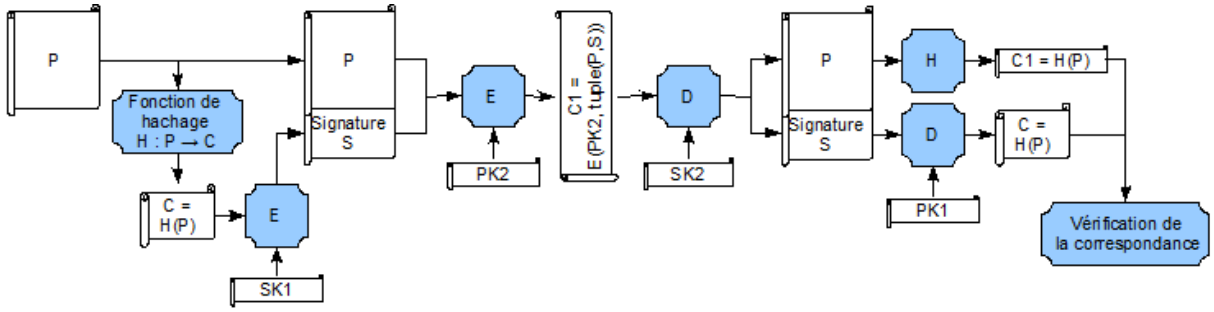


FIG. 2.6 – Authentification par signature

La figure 2.6 présente un mécanisme de signature qui combine une fonction de chiffrement et une fonction de hachage, où l'information à échanger P est signée par deux couples de clefs asymétriques (SK_1, PK_1) et (SK_2, PK_2), resp. du service émetteur et du service destinataire. Le service émetteur génère une empreinte $H(P)$ de l'information à échanger P en appliquant une fonction de hachage $H : P \rightarrow C$ sur P, puis chiffre cette empreinte avec sa clef privée SK_1 afin de produire sa signature numérique S ($S = E(SK_1, H(P))$). Cette signature est attachée à P, puis P et S sont chiffrées par la clef publique du service destinataire PK_2 avant de tout envoyer. Quant au service destinataire, la clef privée SK_2 qu'il détient lui permet de s'authentifier comme le destinataire de l'information échangée, et la clef publique PK_1 permet de vérifier l'authenticité du service émetteur.
La *signature cachée* (blind signature [Cha83, SS95], illustrée dans la partie gauche de la figure la figure 2.7) est un type spécial de signature d'une information sans connaître sa valeur. Elle est utilisée pour signer l'argent numérique ou pour faire signer un tiers de confiance.
La *signature couple* (dual signature [set97, rfcb], illustrée dans la partie droite de la figure la

[7] Les services sont supposés inter-opérables, donc ils ont connaissance de la structure des informations échangées afin de les récupérer après le déchiffrement.

figure 2.7) est un type de signature permettant au destinataire d'une information d'authentifier non seulement l'expéditeur, mais aussi les autres destinataires d'une information reliée. Elle est utilisée en cas d'envoi d'informations reliées à différents destinataires, par exemple des transactions commerciales multi-parties.

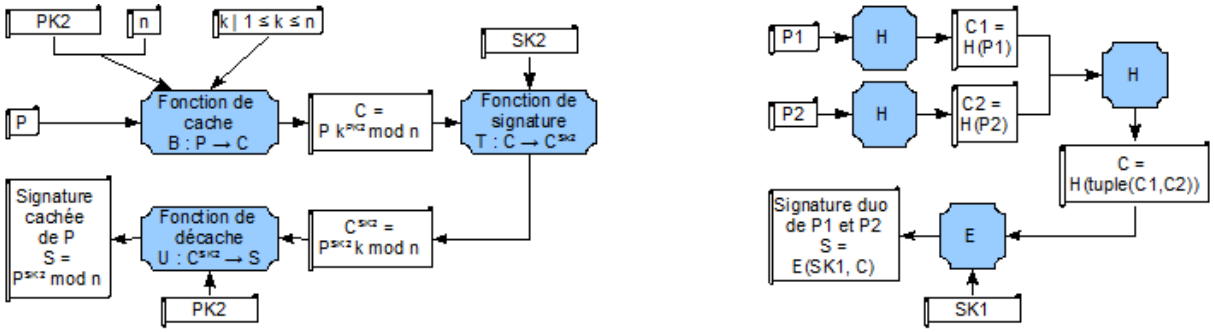


FIG. 2.7 – Signature cachée et signature couple

Hachage Les mécanismes de hachage ont pour but de facilement chiffrer des informations (exploitées, échangées ou stockées) en rendant très difficile, voir impossible de retrouver les informations initiales. Une fonction de hachage $H : P \rightarrow C$ produit l'information chiffrée C dont la taille est à la fois fixée et plus petite que l'information initiale P. C est l'empreinte numérique de P. La fonction de hachage permet de produire deux empreintes différentes correspondant à deux informations initiales P et P_1 différentes.
Par exemple, SHA-1 [sha04] est une fonction de hachage qui est utilisée par le standard de signature numérique (Digital Signature Standard [fip00]). Elle génère l'empreinte des informations échangées sous forme d'un texte chiffré de 160 bits. MD5 [Riv92] est une autre fonction de hachage qui est utilisée par le système d'exploitation Unix pour chiffrer des mots de passe des utilisateurs en forme des textes chiffrés de 128 bits.

Certificat Les mécanismes de chiffrement asymétrique et de signature vérifient l'authenticité et l'origine des informations échangées en se basant sur l'échange des clefs publiques comme preuves d'authentification. Cependant, des clefs publiques échangées peuvent être substituées par des clefs publiques falsifiées. Afin de contrôler cette vulnérabilité, des clefs publiques peuvent être emballées dans des certificats signés qui valident les clefs. Le certificat doit être validé (c.-à-d. , sa signature est vérifiée) avant l'usage de la clef publique qui lui est attribuée.

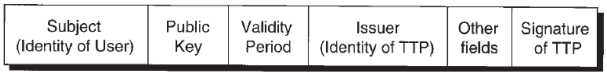


FIG. 2.8 – Structure typique d'un certificat

Un certificat constitue la carte d'identité numérique de l'exploiteur (voir la figure 2.8). Il contient typiquement la clef publique, l'identification de son propriétaire, l'identification de l'organisme qui l'a délivré et une période de validation. Il est signé par une autorité de certification dont les clefs peuvent être certifiées par une autorité de plus haut niveau. Des certificats sont obtenus à partir de l'autorité de certification et ensuite ils sont cachés ou distribués avec les messages qui emploient leurs clefs. Par exemple, X.509 [Ca08] est un service de chiffrement qui permet

de certifier les signatures ou les clefs publiques d'un système de chiffrement asymétrique afin d'authentifier leur propriétaire. Il fournit des fonctions de certification électronique et de validation de chemin de certification. Il est considéré comme un standard de cryptographie de l'Union Internationale des Télécommunications pour les infrastructures à clefs publiques (PKIX [Ca08]).

2.2.1 Authentification et plan de coordination

Dans le cadre du plan de coordination, les besoins génériques d'authentification impliquent :
- l'authentification mutuelle : la mise en œuvre des mécanismes permettant de vérifier l'identité du service appelé ainsi que du service appelant ;
- l'authentification multi-preuves : la mise en œuvre des mécanismes permettant de vérifier plusieurs types de preuves sur un même schéma d'échange ;
- l'authentification multi-schémas : la mise en œuvre des mécanismes permettant de vérifier un type de preuve sur plusieurs schémas d'échange ;
- l'authentification de délégués : la mise en œuvre des mécanismes permettant de vérifier la source d'un appel délégué dans le cadre d'une activité et l'authenticité des activités.

L'authentification réduit les vulnérabilités du plan de coordination qui sont potentiellement exploitées par des attaques de type "*masquerading*" ou "*infiltration*". Une telle authentification reflète les contraintes d'authentification identifiées en vue de l'expression sémantique et de la gestion de l'exécution coordonnée de services. Pour cette raison, l'aspect authentification devrait être adressé en considérant ces deux capacités comportementales des modèles de coordination spécifiques.

En vue de l'expression sémantique de l'exécution coordonnée de services. Les langages de description associés aux modèles d'orchestration (par exemple WS-BPEL [wsb07], OWL-S [owl04]) et aux modèles de chorégraphie (par exemple Linda [CG89], WS-CDL [wsc05], WSCI [wsc02]) permettent de définir le plan de coordination. Ce plan exprime donc la sémantique de l'exécution coordonnée de services. Néanmoins, ces langages ne permettent pas de spécifier l'aspect authentification de l'exécution coordonnée de services lors de la définition du plan de coordination. Il n'y a pas de langage de description associé aux modèles de collaboration, donc il n'est pas non plus possible de le faire dans l'existant.

En vue de la gestion de l'exécution coordonnée de services. Les travaux actuels sur cet aspect sont orientés sur une architecture d'exécution du plan de coordination. Deux besoins d'authentification génériques sont considérées : (i) l'authentification des éléments constitutifs (qui sont tous décrits sous forme de services) de l'architecture qui exécute le plan de coordination ; cette authentification est mise en œuvre en se basant sur les modes d'emploi de ces services ; (ii) l'authentification de la source des messages échangés entre ces éléments au travers des services d'interaction utilisés (communication orientée données ou orientée contrôle). L'authentification est alors définie de manière ad-hoc par rapport aux besoins spécifiques de l'application à base de services, par rapport à chaque modèle de coordination et également par rapport à chaque contexte d'exécution (voir TAB. 2.1). Pour cette raison, les contraintes et les propriétés de l'aspect authentification sont définies vis à vis des éléments constitutifs de l'architecture exécutant le plan de coordination. Par exemple, le langage SAML [sam05] permet de définir les contraintes et les propriétés d'authentification pour les éléments exécutant le plan de coordination des services Web.

	Versions sécurisées du moteur d'exécution du plan de coordination	Configurations sécurisées du moteur d'exécution du plan de coordination	Services de sécurité dédiés/génériques réutilisables par le moteur
Modèles d'orchestration	x		x
Modèles de chorégraphie		x	x
Modèles de collaboration		x	x

TAB. 2.1 – Possibilité de mettre en œuvre des mécanismes d'authentification conformément aux modèles de coordination spécifiques

L'authentification peut être mise en œuvre selon l'une des trois approches mentionnées dans la section 2.1.2. La troisième approche, permettant la mise en œuvre de mécanismes d'authentification sous forme de services dédiés/génériques, est celle qui donne le plus d'avantages. Dans ce cas, les mécanismes de sécurité supportés sont inter-opérables et sont faciles à gérer par l'application à base de services. Néanmoins, les mesures de sécurité associés aux modèles de coordination actuels ne supportent pas l'authentification pour les activités. Les travaux actuels s'appuient sur les mécanismes d'authentification qui sont déjà supportés par les services utilisés (les services à coordonner, les services d'interaction et de réseau de l'infrastructure à communication). Par exemple, l'authentification mutuelle et l'authentification des délégués reviennent à la mise en œuvre du modèle Kerberos [ker02] (qui se base sur des mécanismes de chiffrement) à l'égard des services ; l'authentification multi-preuves et l'authentification multi-schémas revient à la mise en œuvre de modèle Kerberos à l'égard des services et à la mise en œuvre du modèle WS-Security [wss] à l'égard de la communication.

2.2.2 Authentification et communication

Durant la communication entre les services dans le cadre de l'exécution du plan de coordination, il est nécessaire de vérifier l'authenticité de la source des informations échangées et l'authenticité de ceux qui manipulent ces informations (en tant qu'expéditeur, destinataire, propriétaires, etc. des informations échangées). L'authentification à l'égard de communication est considérée à partir de deux points de vue : des services d'interaction dédiés à un modèle de coordination spécifique et des services d'interaction de l'infrastructure.

2.2.2a Services d'interaction génériques ou dédiés à chaque modèle de coordination

La deuxième approche, visant à mettre en œuvre des configurations sécurisées exécutant des plans de coordination, est la plus utilisée pour authentifier ces services. Cette mise en œuvre est liée aux modèles de communication associés aux modèles de coordination : communication orientée contrôle et communication orientée données.

Communication orientée contrôle. Il s'agit de l'authentification autour des informations qui sont échangées sous formes de messages ou d'évènements.

– Authentification dans le cas de l'échange d'informations par des messages : L'idée est d'utiliser différents mécanismes d'authentification pour vérifier l'authenticité de la source des informations intégrées dans les différentes parties d'un message. De cette façon, l'authenticité des services qui peuvent interpréter les informations intégrées dans chaque partie est reconnue. Par exemple, WS-Security Policy [wsp06], WS-Secure Conversation [wss07] permettent de définir des formats étendus pour les messages échangés entre services Web par

le protocole SOAP [soa06]. Cependant, il ne supporte pas l'authentification pour les activités, car il n'y a aucun lien entre une spécification de plan de coordination et le contenu particulier d'un message SOAP.

– Authentification dans le cas de l'échange d'informations par des évènements : L'idée est de construire des services d'évènements spécifiques qui intègrent des mécanismes d'authentification. L'authenticité des services qui les utilisent pour échanger des évènements est alors reconnue. Par exemple, [LLL06] propose une version sécurisée des services d'évènement CORBA pour faire communiquer des composants, [HLP05] propose un canevas orienté évènement pour faire communiquer des services Web.

Communication orientée données. Il s'agit de l'authentification autour de l'espace de données partagé. Avec l'idée de construire un espace de données partagé sécurisé, l'authentification est donc un aspect indispensable pour un tel espace. Les services devant accéder à cet espace doivent fournir la preuve attendue par le mécanisme d'authentification utilisé. Par exemple, [SB03] présente une version sécurisée de service qui gère l'espace de données partagé (*secure tuplespace service*) entre des services. Ces services sont développés suivant un modèle par agents. Il propose également une version sécurisée du protocole d'accès à cet espace qui permet d'authentifier des services qui l'utilisent. [FLZ06] fournit une configuration sécurisée de l'architecture d'exécution pour les plans de coordination de modèle de chorégraphie Linda [CG89]. Il permet de vérifier l'authenticité de la source des données partagées. Cette vérification se base sur des certificats fournis par une autorité de certification intégrée dans le service qui gère l'espace de données partagé.

2.2.2b Services d'interaction et de réseau de l'infrastructure

L'authentification permet de vérifier (durant la communication entre des services) l'authenticité de la source des informations échangées et l'authenticité de ceux qui manipulent ces informations en tant qu'expéditeur, destinataire, propriétaire, etc. des informations échangées. Néanmoins, à cet égard, il est impossible de vérifier l'authenticité des activités ainsi que l'authenticité des fonctions appelées dans le cadre des activités. La raison est qu'il n'y a aucun lien entre une spécification du plan de coordination et la chaîne de services d'interaction et de réseau de l'infrastructure de communication qui est utilisée par les services ; cette chaîne est transparente du point de vue de la spécification du plan.

L'authentification à l'égard de services d'interaction et de réseau de l'infrastructure de communication est mise en œuvre par les trois approches mentionnées dans la section 2.1.2. La première approche, permettant de mettre en œuvre des versions sécurisées des services d'interaction et de réseau de l'infrastructure de communication, est largement utilisée. L'idée est de construire des canaux de communication sécurisés, où les services qui communiquent entre eux peuvent être authentifiés. Pour cela, l'authentification est assurée par des services d'interaction qui s'occupent des aspects transport et interconnexion, par exemple SSL [ssl96, rfcc]. La troisième approche est aussi très répandue, dans le sens où les services de sécurité seront réutilisés pour la mise en œuvre d'autres mécanismes de sécurité de différents aspects, par exemple PKI [SB03].

2.2.3 Authentification et services à exploiter

Dans le contexte d'exécution coordonnée, un service a besoin d'authentifier ses exploiteurs (ceux qui l'accèdent ou ceux qui appellent ses fonctions) en utilisant un (ou plusieurs) mécanisme(s) d'authentification interne ou externe. L'authentification est alors notamment effectuée chez le service appelé. L'authentification peut être effectuée chez le service appelant, puis le service appelé réutilise le résultat de l'authentification. Le service à authentifier doit fournir la preuve attendue par le service d'authentification utilisé (sous forme de propriétés d'authentification). De cette façon les propriétés et/ou les contraintes d'authentification sur une configuration des éléments constitutifs de l'architecture d'exécution sont validées. Afin de faciliter l'exploitation des résultats d'authentification entre des services Web, il existe des technologies et des standard permettant de communiquer entre eux des propriétés et des contraintes d'authentification comme SAML [sam05], WS-Trust [wst00], WS-Adressing [wsa04], AON (Cisco Application-Oriented Networking [cis04]), etc.

Au niveau de services, des mécanismes d'authentification sont mis en œuvre selon l'une des trois approches mentionnées dans la section 2.1.2.

En vue de l'accessibilité des services Les mécanismes servant à l'authentification simple ou mutuelle des exploiteurs supportés par un service sont très liés à son accessibilité. Les mécanismes d'authentification sont mis en œuvre en considérant des différents cas d'appel selon les approches mentionnées ci-dessus. La première approche correspond mieux à la mise en œuvre de l'authentification pour des appels effectués par des services qui possèdent la propriété d'authentification requise. La deuxième approche correspond mieux à la mise en œuvre de l'authentification pour des appels délégués. Par exemple, dans le cas où l'application à base de services appelle un service, cette approche permet d'authentifier le composant qui fait l'appel par sa propre identité ou par l'identité de l'application à base de services. La troisième approche correspond mieux à la mise en œuvre de l'authentification pour des appels entre les services qui ne fournissent pas l'authentification eux-même.

En vue de la flexibilité des services Des mécanismes servant à l'authentification mutuelle et l'authentification multi-schémas des exploiteurs d'un service influencent sa flexibilité. Ils sont mis en œuvre en considérant des différents cas d'exploitation de fonctionnement par des appels aux fonctions spécifiques. La première approche correspond mieux à la mise en œuvre de l'authentification des exploiteurs qui appellent des fonctions permettant d'adapter la configuration de son exécution, ou d'intervenir à son exécution. La troisième approche correspond mieux à la mise en œuvre de l'authentification des exploiteurs qui appellent des fonctions permettant de le paramétrer. La deuxième approche correspond mieux à la mise en œuvre de l'authentification des exploiteurs qui appellent des fonctions permettant de personnaliser la structure de son interface et de son mode d'emploi.

2.3 Autorisation[8]

L'autorisation consiste à décider des droits et des permissions effectives d'accès pour les exploiteurs de services ou de ressources protégées. Il existe deux approches principales pour établir une telle décision : l'une se base sur la combinaison de mécanismes d'authentification

[8] "A process for granting approval to a system entity to access a system resource." [rfca]

et de contrôle d'accès, et l'autre se base sur les mécanismes de gestion de confiance (*trusted management*). L'autorisation a pour but de réduire des vulnérabilités qui sont exploitées par des attaques de type "*denial of service*", "*gathering*" ou "*interception*".

Dans la première approche (voir la figure 2.9), une autorité centralisée commence par vérifier l'identificateur de l'exploiteur par un mécanisme d'authentification. Puis cette autorité centralisée établit la décision par des mécanismes de contrôle d'accès en fonction des critères du profil d'exploiteur authentifié et de ses besoins : l'appel, la manipulation sur un service ou une ressource, les groupes auxquels un service authentifié appartient, ses rôles, etc. Les informations utilisées pour prendre une telle décision sont souvent préalablement reconnues ou enregistrées par cette autorité. De cette façon, l'autorisation est donc très liée à l'authentification. Autrement dit, les mécanismes d'autorisation utilisés devraient être inter-opérables avec les mécanismes d'authentification utilisés.

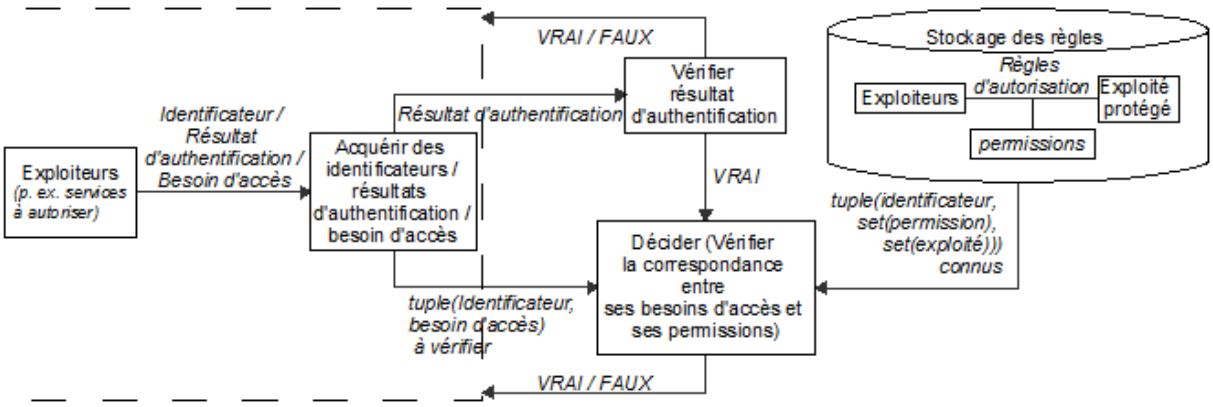


FIG. 2.9 – Autorisation par l'authentification et le contrôle d'accès

- **Contrôle d'accès discrétionnaire** L'idée est d'effectuer la décision d'autorisation au nom du propriétaire de ressources ou du fournisseur de services. Un mécanisme de contrôle d'accès discrétionnaire (DAC) ne contrôle que les droits d'accès d'un service à une ressource. Ce mécanisme est habituellement exprimé sous forme d'une matrice de contrôle d'accès, où les lignes représentent des services, et les colonnes représentent des ressources (voir la figure 2.10). L'intersection d'une ligne et d'une colonne contient le type de droit d'accès du service à la ressource correspondante. Il existe deux façons de mettre en œuvre une telle matrice. La première, dite orientée ligne ou liste des capacités (p.ex. [HR78]), est considérée à partir du point de vue des services à autoriser : pour chaque service, il y a une liste des ressources autorisées et les droits d'accès à ces ressources. La deuxième, dite orientée colonne ou liste de contrôle d'accès (p.ex. [Gla97]), est considérée à partir du point de vue des ressources à protéger : pour chaque ressource il y a une liste de services qui y ont accès à elle et de leurs droits d'accès.
- **Contrôle d'accès non discrétionnaire** L'idée est de contrôler au nom de l'application cible non seulement les droits d'accès d'un service à une ressource, mais aussi les activités que le service peut faire sur cette ressource. MAC (Mandatory Access Control), RBAC (Role-Based Access Control) et TBAC (Task-Based Access Control) sont des modèles populaires de contrôle d'accès non discrétionnaire.

  MAC : Dans ce modèle, une autorité centralisée de prise de décision détermine quelle

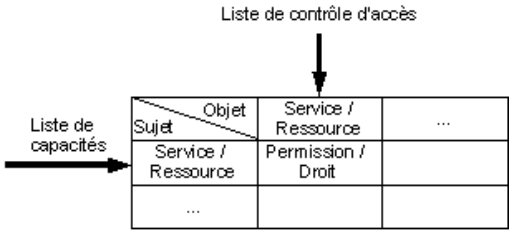


FIG. 2.10 – Discretionary Access Control

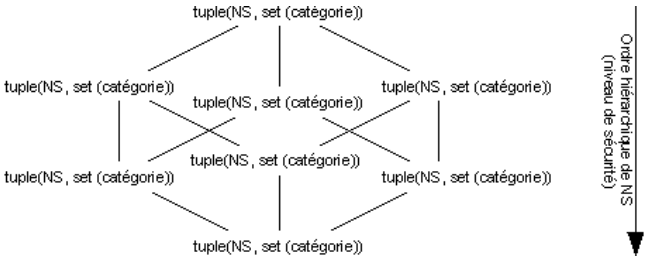


FIG. 2.11 – Lattice-Based Access Control

ressource est accessible par quel service. Les propriétaires de ressource n'ont pas le droit de contrôler l'accès à la ressource. Ce modèle se base sur un nombre restreint d'étiquettes d'autorisation qui sont universellement imposés aux services et ressources à protéger. Les étiquettes de services spécifient leurs niveaux de confiance, celles de ressource spécifient des niveaux de confiance requis pour l'accès. De cette façon, l'envoi et la réception d'informations par les services ou les ressources sont aussi contrôlés. Il existe deux façons de mettre en œuvre de MAC : Rule-Based Access Controls (utilisé souvent pour le contrôle des flux des informations entre deux services ou ressources, p.ex. [Bib77]) et Lattice-Based Access Controls (utilisé souvent pour le contrôle des flux des informations entre plusieurs services ou ressources [Den76, BL75], voir la figure 2.11).

RBAC : En utilisant un modèle RBAC (p.ex. [DF03, SFY96]), des permissions pour des

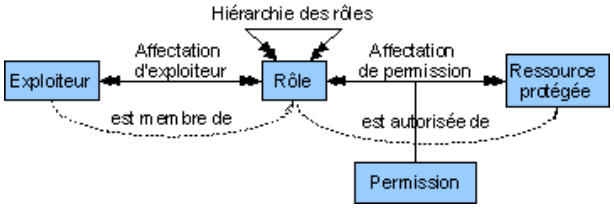


FIG. 2.12 – RBAC

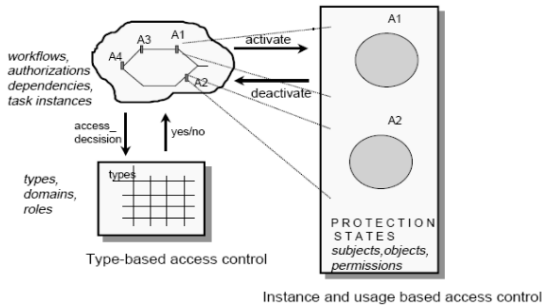


FIG. 2.13 – TBAC [TS97]

manipulations sur des ressources constituent un rôle du service. Les fonctions dont l'accès est à restreindre sont variées ; il s'agit de fonctions de manipulation complexes de l'application à base de services ou des fonctions de manipulation simple des services d'interaction (lecture, ouverture, etc.). Le droit d'accès est donc affecté en se basant sur le rôle du service. Pour cette raison, le modèle RBAC rend l'autorisation à la fois plus flexible et plus contrôlable que les modèles DAC ou MAC présentés ci-dessus. Il est largement utilisé en vue des besoins d'autorisation des applications distribuées.

TBAC : Un modèle TBAC (p.ex. [TS97]) permet de définir des permissions d'accès du point de vue des tâches à accomplir conformément à une logique applicative concrète. Il convient bien d'être intégré dans des moteurs d'orchestration, par exemple un gestionnaire de workflow qui supporte l'autorisation.

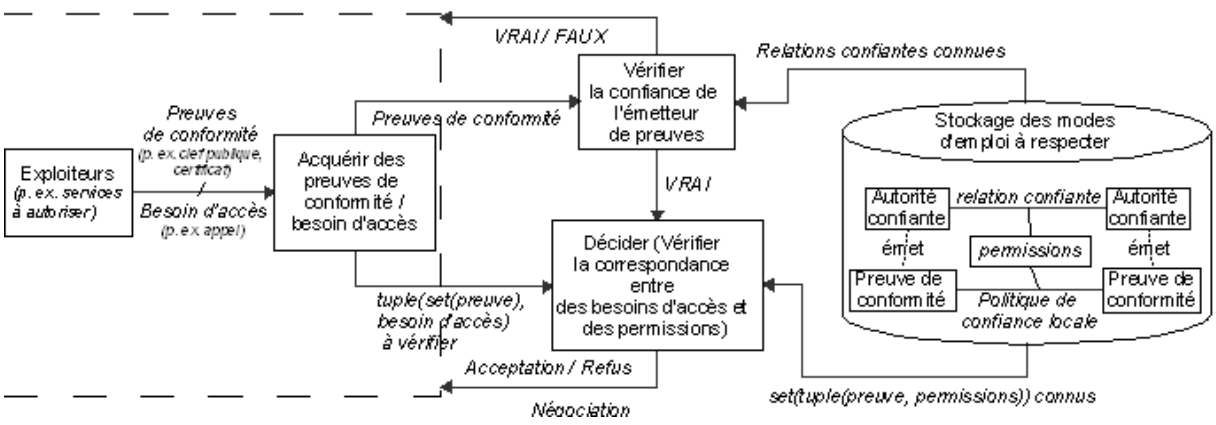


FIG. 2.14 – Autorisation par la gestion de confiance

Dans la deuxième approche (voir la figure 2.14), les droits et les permissions effectives d'accès (à un service) d'un exploiteur ne sont pas associés aux identificateurs des exploiteurs. Pour cette raison, elle permet de prendre la décision d'autorisation d'une manière plus efficace pour les applications distribuées, surtout pour des appels effectués par des exploiteurs dont l'identificateur est inconnu au préalable par l'autorité qui prend la décision d'autorisation.

La décision de droits et de permissions pour un exploiteur est établie par une autorité en se basant sur des preuves de conformité (*credentials*) qui sont présentées par l'exploiteur. Une preuve de conformité démontre non seulement l'identité de l'exploiteur, mais aussi des informations sur ses droits et permissions effectives d'accès qui ont été décidées et/ou certifiées par d'autre(s) autorité(s). Autrement dit, les droits et permissions accordées par une autorité deviennent une preuve de conformité pour l'exploiteur. Par exemple qu'un certificat délivré par une autorité de confiance (voir la section 3.3.4) ou qu'une clef publique certifiée par une autorité de confiance.

Une autorité établit sa décision propre en fonction du mode d'emploi de ses ressources ainsi que des conventions préalables de confiance entre elle et autres autorités concernées. Cela dépend de la façon dont les preuves de conformité sont démontrées, ainsi que des règles de gestion de confiance spécifiques. La gestion de confiance permet de décentraliser l'autorité qui prend la décision d'autorisation au nom de service à exploiter. D'une part, une autorité peut devenir un procureur qui établit la décision au nom d'autre(s) autorité(s) lui faisant confiance. D'autre part, une autorité peut réutiliser la décision établie par d'autre(s) autorité(s) à laquelle (auxquelles) elle fait confiance. De cette façon, l'autorisation se ramène à la vérification des relations de confiance entre des autorités déléguées (authentification des autorités déléguées) et la vérification de la correspondance légitime entre des preuves de conformité et des modes d'emploi prédéfinis de service.

2.3.1 Autorisation et plan de coordination

L'autorisation à l'égard du plan de coordination concerne d'une part la restriction de l'exploitation de l'application à base de services par des exploiteurs autorisés. D'autre part, cela implique que l'application à base de services a suffisamment de droits et de permissions pour l'exécution des activités du plan de coordination.

Du point de vue de l'application à base de services, chaque activité dans le plan d'exécution consiste en l'exécution de plusieurs fonctions des services constitutifs (service appelant, service

appelé, tiers de confiance, vérificateurs, etc). Dans le cas idéal, l'application à base de services devrait vérifier les règles de contrôle d'accès de tous les services constitutifs avant de démarrer une activité. Cela requiert que toute règle de contrôle d'accès de services constitutifs soit vérifiable par l'application (par exemple par une autorité commune qui prend la décision au nom de l'application à base de services). En pratique, il est presque impossible de faire cela, car des règles de contrôle d'accès sont établies par chaque service constitutif de manière opaque. Par ailleurs, il n'est pas pertinent de codifier une stratégie spécifique de contrôle d'accès aux éléments constitutifs de l'architecture d'exécution d'un service : la réutilisabilité du service peut être fortement réduite dans le cas où l'exploiteur de service requerrait un modèle différent de contrôle d'accès. Afin de supporter l'autorisation en termes des mécanismes de contrôle d'accès à l'égard de l'application à base de services, il est nécessaire de considérer (i) le niveau sur lequel ils sont appliqués : à l'égard de services constitutifs ou à l'égard de communication ; (ii) la force des mécanismes utilisés pour renforcer le contrôle d'accès ; (iii) le niveau de l'intégration des mécanismes de contrôle d'accès dans des mécanismes de gestion du plan de coordination.

En vue de l'expression sémantique de l'exécution coordonnée de services. Les modèles de coordination actuels ne permettent pas de spécifier l'aspect autorisation de l'exécution coordonnée de services lors de la définition du plan de coordination. Néanmoins, certains travaux sur des modèles de contrôle d'accès tentent d'adresser l'aspect autorisation vis-à-vis des activités spécifiques d'un plan de coordination. Par exemple, le modèle TBAC [TS97] permet d'adresser l'aspect autorisation vis-à-vis des tâches spécifiques à exécuter (par exemple un appel) d'un workflow, mais non vis-à-vis de l'expression sémantique de l'exécution coordonnée de services. [BFA99] est une variante du modèle RBAC qui permet de définir (en terme des contraintes) des rôles et des utilisateurs spécifiques pour des agents qui exécutent les tâches d'un workflow.

En vue de la gestion de l'exécution coordonnée de services. Les travaux actuels sur cet aspect sont orientés sur une architecture d'exécution du plan de coordination. L'idée est de favoriser le partage et/ou l'échange d'informations entre les éléments constitutifs de cette architecture. Ces travaux se ramènent alors aux mécanismes d'autorisation appliqués à l'égard de la communication. L'autorisation est alors ad-hoc non seulement par rapport aux besoins spécifiques de l'application à base de services, mais aussi par rapport à chaque modèle de coordination ainsi qu'à chaque contexte d'exécution de la communication. Tab. 2.2 résume les possibilités de mise en œuvre d'une telle autorisation selon l'une des trois approches permettant de sécuriser le moteur de coordination. Par exemple, [EDS02] propose une mise en œuvre (resp. la version

	Versions sécurisées du moteur d'exécution du plan de coordination	Configurations sécurisées du moteur d'exécution du plan de coordination	Services de sécurité dédiés/génériques réutilisables par le moteur
Modèles d'orchestration	x		
Modèles de chorégraphie			x
Modèles de collaboration		x	x

Tab. 2.2 – Possibilité de mettre en œuvre des mécanismes d'autorisation conformément aux modèles de coordination spécifiques

sécurisée du moteur) du modèle de contrôle d'accès RBAC pour un moteur de workflow existant (resp. du modèle d'orchestration). [HK03b] propose un moteur de workflow multi-couche permettant de gérer et de surveiller le flux d'autorisation durant l'exécution des taches d'un

workflow (resp. la version sécurisée du moteur). [HKR05] propose une approche parcellaire pour contrôler l'aspect autorisation des éléments constitutifs d'un moteur de collaboration qui est exécuté dans l'environnement VO (*Virtual Organisations*).

2.3.2 Autorisation et communication

À cet égard, l'autorisation est considérée à partir de deux points de vue : des services d'interaction dédiés à un modèle de coordination spécifique et aux services d'interaction de l'infrastructure.

2.3.2a Services d'interaction génériques ou dédiés à chaque modèle de coordination

L'autorisation à cet égard favorise le partage et l'échange optimal des informations. Néanmoins, comme l'autorisation est spécifiquement établie pour chaque application à base de services, cela devrait être lié aux modèles de coordination, en termes des modèles de communication sécurisés utilisés par chaque modèle de coordination. Les mécanismes d'autorisation sont souvent mis en œuvre sous forme des services de sécurité dédiés ; cela permet la réutilisation des résultats d'authentification d'un service ou d'une ressource pour décider son autorisation.

Communication orientée contrôle. Il s'agit de l'autorisation autour de l'envoi et la réception de messages ou d'évènements.
Dans le cas de l'échange d'informations par messages, l'idée est d'ajouter des propriétés et des contraintes d'autorisation au contenu de messages échangés. De cette façon, une fois qu'un service ou une ressource est authentifié, ses permissions et ses droits sont aussi reconnus. Par exemple, SAML [sam05] permettent de définir des décisions d'autorisation, des propriétés d'autorisation qui sont intégrées au contenu des messages échangés entre des services Web par le protocole SOAP [soa06]. De même, des preuves de conformité qui démontrent directement les droits et les permissions d'un service exploiteur sont également intégrées au contenu de messages échangés comme des propriétés d'autorisation. Plusieurs travaux [EDS02, SK02] s'appuient sur la restructuration des messages XML échangés par le protocoles SOAP avec des propriétés d'autorisation qui sont pré-définies pour des appels dans les contextes d'exécution concrets. Par exemple, des informations sur des liens, sur des emplacements autorisés ou sur des modes d'autorisation utilisés sont incluses dans le contenu des messages échangés. D'autres travaux s'appuient sur les protocoles de négociations au préalable des preuves de conformité et des contraintes d'autorisation [YWS03, RZN+05], ainsi que la gestion des relations de confiance entre des services [BS02, wst00, wsf06].
Dans le cas de l'échange d'informations par évènements, l'idée est de construire des services d'évènement spécifiques qui intègrent à la fois des mécanismes d'authentification et d'autorisation. Par exemple, [BGH+95, BGH+00] proposent la famille de protocole *i*KP supportant les transactions entre le client (un service navigateur) et le vendeur (un service Web) avec l'authentification et l'autorisation de paiement par carte bancaire.

Communication orientée données. Il s'agit du contrôle d'accès à l'espace de données partagé, c.-à-d. la vérification des droits et des permissions pour appeler quatre groupes de fonction : ouverture, lecture, écriture et fermeture. Par ailleurs, la construction d'un espace de données partagé où les données sont regroupées selon leurs permissions est aussi abordée. Des mécanismes

d'autorisation sont souvent mis en œuvre sous forme des services dédiés (selon l'approche parcellaire) ; cela permet des services applicatifs de reconnaître leurs droits et leurs permissions. Par exemple, une infrastructure de clefs publiques telle que [WO99] permet l'autorisation à base de certificats qu'elle supporte, dans le sens où les propriétés d'autorisation ont été incluses dans les certificats échangés. Une telle autorisation est utilisée comme le complément d'un modèle de contrôle d'accès supporté au niveau de services. [SB03] propose également une version sécurisée du protocole d'accès à cet espace qui permet d'authentifier et contrôler des services qui l'utilisent.

2.3.2b Services d'interaction et de réseau de l'infrastructure

L'autorisation à l'égard de l'infrastructure de communication consiste en la vérification des droits d'accès aux services d'interaction et de réseau, c.-à-d. des droits pour établir une connexion par le biais de ces services. Il s'agit de la décision prise si la demande de connexion (appel à une fonction qui met en œuvre la connexion) est acceptée. L'autorisation est effectuée dès l'établissement de la connexion.

Une telle décision est basée sur des propriétés d'autorisation de la ressource protégée ; elle correspond à une fonction d'un service d'interaction ou d'un service de réseau mettant en œuvre la connexion. Cette décision est basée également sur des propriétés d'authentification du service appelant contenues dans l'appel. L'autorisation est liée à la communication dans le sens où l'accès aux services d'interaction, ainsi que l'accès aux services de réseau sur des aspects transport et interconnexion de l'infrastructure de communication doit être contrôlé. Dans ce cas, l'autorisation ramène aux mécanismes d'autorisation au niveau des services.

Des mécanismes de contrôle d'accès sont souvent mis en œuvre selon l'approche centralisée. Il s'agit de versions sécurisées de services implantant des protocoles de communication. Elles supportent à la fois l'authentification et l'autorisation. Par exemple, S/MIME [rfc04] est la version sécurisée du protocole MIME [rfc93], qui est utilisé pour échanger des emails ; S-HTTP [rfc99b] est la version sécurisé du protocole HTTP [FGM+06], ATM [CSV96], etc. L'approche décentralisée est moins utilisée, car un mécanisme de contrôle d'accès, qui prend la décision en se basant sur les propriétés d'authentification et des propriétés d'autorisation, n'est pas souvent défini dans la spécification du protocole qui est mis en œuvre par un service d'interaction ou de réseau.

2.3.3 Autorisation et services à exploiter

Dans le contexte de l'exécution coordonnée, un service a besoin de déterminer si son exploiteur authentifié a l'autorisation requise pour faire exécuter une de ses fonctions ou pour révéler ses propriétés ou ses contraintes. Dans ce sens, une autorisation rigoureuse permet de protéger le service contre des attaques. Les mécanismes d'autorisation peuvent être supportés par des services appelants (exploiteurs) ou des services appelés.

- Chez des services appelés, il existe trois possibilités de mise en œuvre des mécanismes d'autorisation :
  1. Il n'y a pas de vérification de l'autorité pour faire exécuter une fonction ou pour revéler des propriétés ou des contraintes.
  2. Seulement des appels de la part du même service sont autorisés (l'identificateur du service appelant est l'identificateur du service appelé).

3. N'importe quel mécanisme d'autorisation qui permet de vérifier l'autorité des exploiteurs à partir de leur identificateurs.

– Chez des services appelants, les mécanismes d'autorisation suivants sont possiblement mis en œuvre :

1. Il n'y a pas de vérification de l'autorité requise pour exécuter un appel.

2. Un appel est autorisé si l'identificateur du service appelé est l'identificateur du service appelant.

3. Un appel est autorisé si l'identificateur du service appelé est reconnu.

L'autorisation peut être mise en œuvre par les trois approches : parcellaire, centralisée et décentralisée. La décision de l'autorisation est souvent propre au service, donc l'approche parcellaire est la plus pertinente. L'approche décentralisée est utilisée dans la plupart des cas, car des informations utilisées comme l'entrée pour la décision d'autorisation sont souvent tirées d'un serveur d'annuaire ou d'un dépôt de configuration de l'architecture d'exécution.

En vue de l'accessibilité de services. La mise en œuvre de services sous différentes formes (service Web, service ouvert, composant, agent, etc.) permet de rendre accessible totalement ou partiellement des fonctions, des propriétés et des contraintes d'un service. Seulement les fonctions décrites dans l'interface d'un service peuvent être accessibles à l'extérieur par des exploiteurs de ce service. C'est le modèle de développement de services qui rend une fonction accessible à travers une ou plusieurs interfaces.
Le modèle d'autorisation d'un service définit en plus des contraintes d'accès aux fonctions décrites dans son interface et les propriétés d'autorisation requises pour un tel accès. Ces contraintes et propriétés sont définies pour une configuration spécifique d'une architecture d'exécution du service lui-même. La mise en œuvre de modèle d'autorisation est ad-hoc ; de cette façon, la décision d'autorisation est souvent propre au service. Les mécanismes de contrôle d'accès non discrétionnaire tel que RBAC sont souvent utilisés à l'égard de services pour contrôler non seulement les droits d'accès d'un service (en tant qu'exploiteur) à une ressource, mais aussi les activités que ce service peut faire sur cette ressource.

En vue de la flexibilité de services. En pratique, la mise en œuvre des services sous différentes formes entraîne de deux possibilités d'accès aux propriétés et contraintes d'un service : soit le service exporte des fonctions permettant d'accéder à ses contraintes en retenant ses propriétés (les propriétés restent inaccessibles), soit le service exporte des fonctions permettant d'accéder à ses propriétés en respectant ses contraintes (les contraintes restent inaccessibles).
Dans la plupart des cas, l'accès aux propriétés non-fonctionnelles et à la plupart des contraintes non-fonctionnelles est interdit par le modèle d'autorisation associé au service.

2.4 Intégrité [9]

L'intégrité consiste en l'affirmation qu'aucun exploiteur n'a causé de changement non autorisé sur des services, ressources ou informations à protéger. Des mécanismes d'intégrité ont été proposés selon deux directions : empêcher ou détecter des modifications, altérations ou destructions non autorisées causées par des exploiteurs (voir la figure 2.15). De cette façon, ils permettent de protéger contre des attaques de types "*tampering*", "*replaying*", des activités ou services non autorisés causés par des exploiteurs (par exemple, des chevaux de Troie, des virus). Les mécanismes d'intégrité sont très liés aux mécanismes d'authentification et d'autorisation, dans le sens où les propriétés d'authentification et d'autorisation (d'exploiteurs, de services, de ressources, d'information) sont nécessaires pour une telle affirmation. Ces propriétés sont également considérées comme des propriétés d'intégrité.

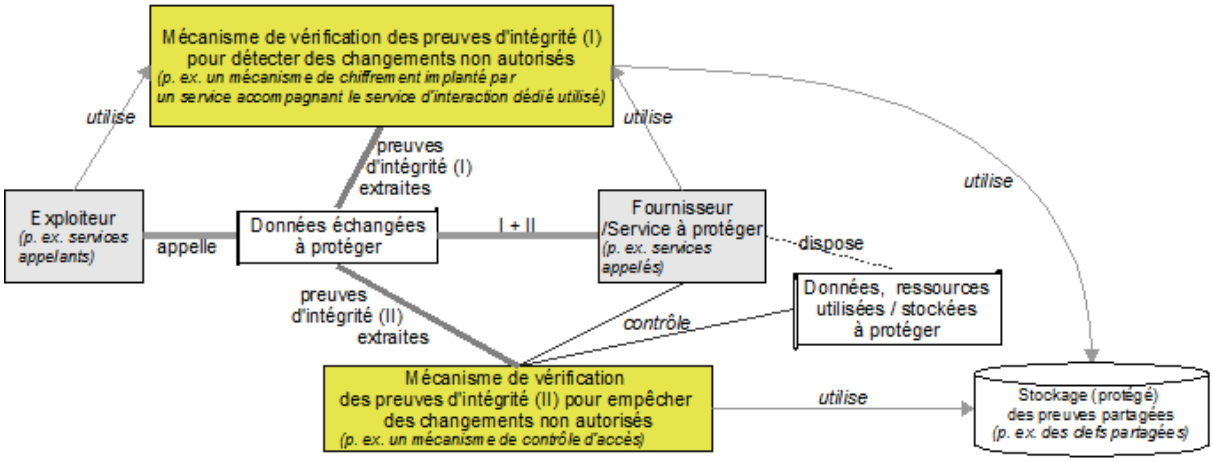


FIG. 2.15 – Principe d'intégrité

Interdiction des changements non autorisés. Les mécanismes de contrôle d'accès supportés par certains modèles MAC (par exemple [Bib77], [LDS+90], etc., voir la section 2.3) assurent l'intégrité en empêchant le changements non autorisé des ressources. L'idée est de protéger les ressources contre (i) des changements effectués par des exploiteurs non autorisés (vérification de droits des exploiteurs) ; (ii) des changements non autorisés effectués par des exploiteurs autorisés (vérification de droits d'exploiteurs et de permissions de ressources).

Par ailleurs, certains mécanismes de contrôle d'accès à base du modèle RBAC [CW87], ou bien des mécanismes de gestion des confiances permettent également d'assurer l'intégrité des données.

Détection des changements non autorisés. D'autres mécanismes permettent de détecter des changements non autorisés : des mécanismes de chiffrement (voir la section 2.2), par exemple en utilisant des signatures numériques sur des messages dont le contenu est chiffré par des fonctions de hachage ; ou bien des codes d'authentification de messages ou des *timestamps*.

[9] "Data integrity : The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner.
System integrity : The process of restoring an information system's assets and operation following damage or destruction.
Source integrity : The property that data is trustworthy (i.e., worthy of reliance or trust), based on the trustworthiness of its sources and the trustworthiness of any procedures used for handling data in the system." [rfca]

Par ailleurs, il s'agit de mécanismes de *checksums*, des mécanismes de la vérification par redondance cycliques, des moyens de protection contre les virus, les vers ou les chevaux de Troie, etc. Ceux-ci permettent aussi de se prémunir contre des changements non autorisés en ayant la certitude que des ressources (services ou informations) n'ont pas été modifiées lors de leur exploitation. Ces mécanismes peuvent être appliqués ensemble ou de façon séparée.

Mécanismes de chiffrements symétriques ou asymétriques ([x9598, fip01], etc.) : les modifications sont détectées suite à l'impossibilité de déchiffrement des informations récupérées. Néanmoins, ils n'assurent pas que des informations n'ont pas été complètement détruites.

Hachage (voir la figure 2.16) : l'empreinte numérique des informations (produite par une fonction hachage de sens unique - digest function) est chiffrée par la clef privée de l'émetteur et associée à ces informations avant de tous envoyer. Lors de la réception, le destinataire le déchiffre avec la clé publique de l'émetteur puis, recalcule l'empreinte à partir des informations récupérées avec la même fonction de hachage. La différence entre l'empreinte numérique récupérée et l'empreinte recalculée prouve que les informations échangées sont altérées.

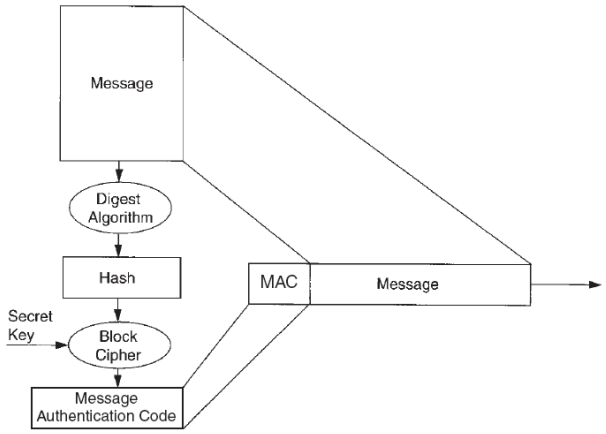


FIG. 2.16 – Empreinte numérique d'un message (Message Authentication Code)[OPTD01]

Checksums : ce mécanisme vérifie si les informations n'ont pas été modifiées. Un résumé (le résultat (éventuellement chiffré) d'une fonction de calcul appliquée à ces informations) y est associé avant de les stocker ou les envoyer. Lors de l'exploitation ou la réception de ces informations, la valeur du résumé est recalculée (ou déchiffrée puis recalculée) avec la même fonction à partir des informations récupérées. Si la valeur obtenue diffère, il en déduit que les informations ont été modifiées.

Vérification par la redondance cyclique : il s'agit une fonction qui permet de transformer des informations en des informations chiffrées de taille fixe. Cette fonction peut être utilisée par le mécanisme de check sums.

2.4.1 Intégrité et plan de coordination

L'intégrité à l'égard de la coordination implique que des informations de l'exécution coordonnée (qu'elles soient exploitées, stockées ou échangées), des activités et des services ne sont pas modifiées, altérées ou détruites d'une manière imprévue ou accidentelle. En vue de la gestion de l'exécution coordonnée des services, les travaux actuels sur cet aspect sont orientés sur une architecture d'exécution du plan de coordination. Des besoins d'intégrité sont considérés :

1. En tant que service, l'intégrité du plan de coordination devrait être assurée.
2. En tant qu'exploiteur des services, particulièrement l'exploiteur des informations que ces services fournissent, des éléments constitutifs de l'architecture d'exécution du plan de coordination ne peuvent modifier d'autres éléments de la même architecture d'une manière qui pourrait les corrompre ou détruire leurs informations durant l'exécution de ce plan. Ils ne peuvent non plus rendre l'information de prise de décision incertaine pour leur plan de coordination.

Les mécanismes assurant l'intégrité utilisés sont alors ad-hoc ; ils peuvent être mis en œuvre selon l'une des trois approches : parcellaire, centralisée ou décentralisée comme illustrées dans Tab. 2.3.

	Versions sécurisées du moteur d'exécution du plan de coordination	Configurations sécurisées du moteur d'exécution du plan de coordination	Services de sécurité dédiés/génériques réutilisables par le moteur
Modèles d'orchestration	x		x
Modèles de chorégraphie			x
Modèles de collaboration		x	x

Tab. 2.3 – Possibilité de mettre en œuvre des mécanismes d'intégrité conformément aux modèles de coordination spécifiques

L'élément verrouillé assurant l'intégrité à ce niveau sont les instructions de contrôle qui définissent strictement la séparation appropriée des fonctionnements ainsi que la vérification et la validation de n'importe quel changement des services ou de leurs fonctions. Les travaux actuels sur cet aspect se ramènent aux modèles d'intégrité à l'égard de la communication et à l'égard de services utilisés. Par ailleurs, certains modèles permettent la description architecturale des mécanismes d'intégrité utilisés en vue de l'architecture d'exécution du plan de coordination, par exemple WS-Policy [wsp06]. Aucun modèle ne permet la description (sémantique ou architecturale) de ces mécanismes en vue de l'expression sémantique de l'exécution coordonnée de services.

2.4.2 Intégrité et communication

Un service d'intégrité permet aux services concernés de vérifier l'intégrité des données échangées par le biais d'une infrastructure de communication. De cette façon, il protège les données contre des écoutes actives qui peuvent modifier les données interceptées et des attaques de type "*tampering*" ou "*replaying*".

2.4.2a Services d'interaction génériques ou dédiés à chaque modèle de coordination

Il s'agit de mécanismes qui assurent l'intégrité des messages échangés entre les éléments constitutifs d'une architecture d'exécution du plan, ainsi que ceux qui visent à l'inter-opérabilité entre des mécanismes d'intégrité utilisés par des services.

Communication orientée contrôle. L'intégrité est adressée autour des transactions entre les services ; il s'agit de mécanismes d'intégrité dite non orientée communication. Ces services dédiés accompagnent des services qui communiquent des messages dans le cadre du plan de coordination. Leurs fonctions permettent de détecter des changements non autorisés des messages envoyés ou reçus dans le cadre d'une transaction spécifique à travers une chaîne de services de

l'infrastructure de communication en commune qui est dynamiquement déterminée. Par exemple, des signatures XML [rfc02] ou des codes d'authentification de messages [rfc03] sont utilisées pour certifier et assurer l'intégrité de messages échangés eux-même entre les services Web. La vérification de signatures ou de codes d'authentification de messages permet de détecter les changements des messages certifiés.
Par ailleurs, il s'agit de mécanismes d'intégrité dite sélective. Ils permettent d'assurer l'intégrité de certaines parties de messages échangés selon les besoins de services qui les appellent. Par exemple, les propriétés de sécurité attachées aux messages échangés (par exemple security token) sont signées pour assurer leur intégrité.

Communication orientée données. L'intégrité est souvent assurée par des mécanismes de contrôle d'accès. L'idée est de contrôler des services qui manipulent des données dans l'espace de données partagé en tant que propriétaire et exploiteur ([SB03]) ou en se basant sur la séparation de leur fonctionnement ([JSSB97]). D'ailleur, l'intégrité est adressée autour du service qui gère les données partagées ; cela se ramène aux mécanismes d'intégrité supportés par des services de gestion de données.

2.4.2b Services d'interaction et de réseau de l'infrastructure de communication

Il s'agit de mécanismes d'intégrité dite orientée communication ; ces mécanismes sont souvent mis en œuvre selon l'une des trois approches : parcellaire, centralisée ou décentralisée. Ces mécanismes accompagnent ou sont codifiés dans des services d'interaction ou de réseau de l'infrastructure de communication. Ils permettent d'assurer l'intégrité de tous les messages échangés par le biais de ces services : leurs fonctionnements principaux sont de détecter et notifier des messages échangées qui sont perdus, reproduits ou désordonnés. Par exemple, il existe plusieurs versions sécurisées des services d'interaction et de réseau, particulièrement ceux qui servent aux aspects transport et interconnexion. Par ailleurs, il s'agit aussi des services dédiés ou génériques mettant œuvre des mécanismes d'intégrité sélective. Ces derniers assurent l'intégrité de certaines parties de messages échangés selon les besoins spécifiques des services qui les appellent. Par exemple, une infrastructure des clefs publiques utilisées à chiffrer une partie qui contient des informations de contrôle du message échangé.

2.4.3 Intégrité et services à exploiter

L'intégrité d'un service peut être protégée en mettant en œuvre des mécanismes d'intégrité présentés ci-dessus : un contrôle d'accès rigoureux ; un chiffrement des données, etc. Par ailleurs, des moyens de protection contre les virus, les vers ou les chevaux de Troie au niveau de la plate-forme d'exécution et au niveau des ressources utilisées (par exemple bases de données) renforcent l'intégrité de service. Néanmoins, des mécanismes d'intégrité mis en œuvre pour des services influencent leur disponibilité, leur accessibilité et leur flexibilité.

En vue de la disponibilité de services. La disponibilité représente la capacité d'un service à mener correctement à terme ses fonctions, ses propriétés et ses contraintes. D'une part, elle dépend de la mise en œuvre du service : composant, service Web, service ouvert, agent, etc. D'autre part, elle dépend de modèles de qualité de services associés tels que sauvegarde, performance, ergonomie, maintenance opérationnelle, tolérance aux pannes, sûreté de fonctionnement, sécurité, etc. Par exemple, un service mis en œuvre sous forme d'un agent peut refuser un appel

pour certaines raisons comme la surcharge d'une autre fonction. L'intégrité permet en plus que le service est en état attendu par ses exploiteurs, dans le sens où les mécanismes d'intégrité permettent aux exploiteurs autorisés d'appeler les fonctions dont ils ont besoin. Il s'agit de la mise en œuvre de manière cohérente de plusieurs mécanismes d'intégrité présentés ci-dessus, en vue des besoins d'intégrité spéciales de chaque service à protéger.

En vue de l'accessibilité de services. Les mécanismes d'intégrité influencent plus ou moins la capacité d'un service à répondre à un appel à une de ses fonctions en retenant ses propriétés et en respectant ses contraintes. Ils assurent l'intégrité des fonctions, propriétés et contraintes du service à travers le contrôle d'accès. Par ailleurs, le modèle d'intégrité est défini non seulement en vue de la description architecturale d'un modèle d'instanciation de service spécifique, mais aussi en vue du modèle de communication associé à cette architecture.

En vue de la flexibilité de services. Du point de vue de la sécurité, l'exécution des fonctions de personnalisation, d'adaptation et de paramétrisation devrait assurer l'intégrité du service. Pour cette raison, les propriétés non-fonctionnelles et la plupart des contraintes non-fonctionnelles restent non modifiables. Certains services fournissent des fonctions permettant d'adapter la configuration de l'architecture d'exécution en retenant des propriétés et/ou en satisfaisant des contraintes du service. Par exemple, [SV00] définit des adaptateurs qui sont capables d'encapsuler des services existants et de les exécuter en retenant des propriétés de sécurité de services.

2.5 Confidentialité[10]

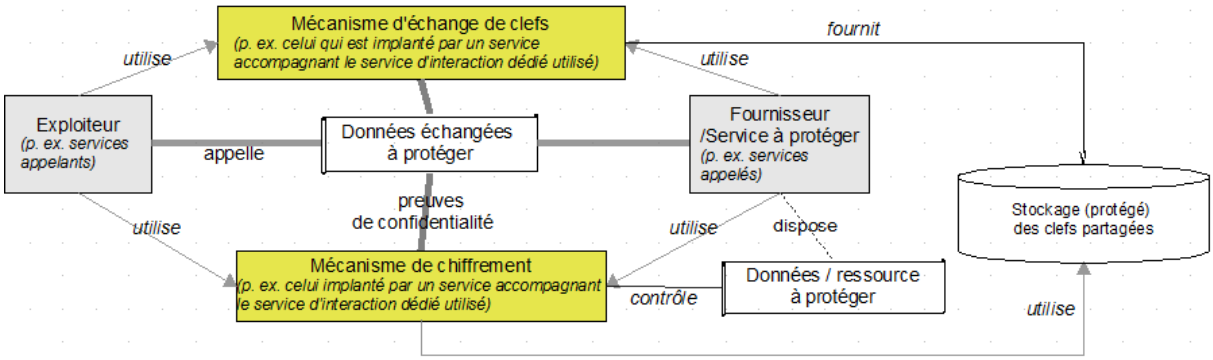


FIG. 2.17 – Principe de confidentialité

Les besoins de confidentialité sont spécifiquement reliées aux informations. La confidentialité implique que toute information sensible n'est rendue compréhensible que pour les exploiteurs prévus. Par ailleurs, il implique aussi l'intimité numérique : la propagation des informations appartenant en propre aux services ou définissant l'exécution coordonnée est contrôlée (quand, comment et à quel degré de sensibilité la propagation est autorisée aux exploiteurs).

Des mécanismes assurant la confidentialité des informations consistent principalement en le contrôle d'accès à ces informations tel que [BL75] (voir la section 2.3) et le chiffrement/déchiffrement des informations (voir la figure 2.17).

[10] "The property that data is not disclosed to system entities unless they have been authorized to know the data." [rfca]

Signature Contrairement à un mécanisme de chiffrement pour l'authentification, un mécanisme de chiffrement pour la confidentialité permet de chiffrer l'information initiale par une clef publique et la déchiffrer par la clef privée correspondante. De cette façon, l'information initiale n'est révélée qu'auprès du service destinataire qui détient la clef privée (voir la figure 2.18).

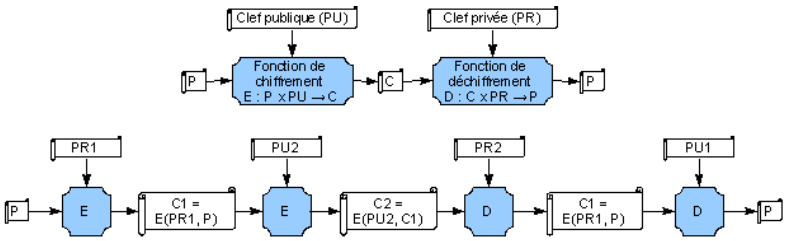


FIG. 2.18 – Confidentialité par la signature

Enveloppe Par ailleurs, la combinaison de mécanismes de chiffrement symétrique et asymétrique permet de produire une enveloppe assurant la confidentialité pour l'information échangée (voir la figure 2.19). Le service émetteur génère aléatoirement une clef symétrique K, puis chiffre l'information échangée (P) par une fonction de chiffrement avec cette clef ($E : P \times K \rightarrow C$). Cette clef symétrique est également chiffrée par la clef publique du service destinataire et jointe à C avant de tout envoyer. De cette façon, l'information initiale n'est révélée qu'auprès du service destinataire qui détient la clef privée permettant de déchiffrer la clef symétrique utilisée pour chiffrer l'information initiale.

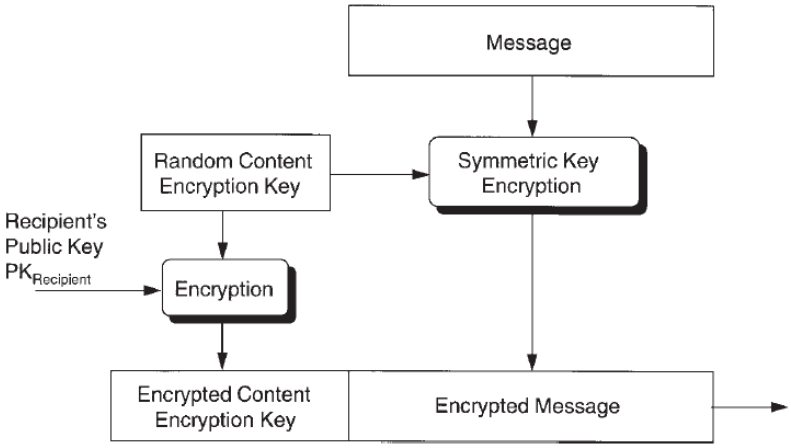


FIG. 2.19 – Confidentialité par l'enveloppe[OPTD01]

De cette façon, ils protègent une application à base de services contre des attaques de types "*eavesdropping*", "*gathering*".

2.5.1 Confidentialité et plan de coordination

En vue de l'expression sémantique de l'exécution coordonnée de services. Il n'y a aucun besoin spécifique de confidentialité ou d'intimité numérique ; cela est issu des besoins d'authentification, d'autorisation et d'intégrité.

En vue de la gestion de l'exécution coordonnée de services. La confidentialité et l'intimité consistent en la protection d'information de l'exécution coordonnée de sorte que les

exploiteurs (les éléments constitutifs de l'architecture d'exécution du plan de coordination) non autorisés ne puissent pas y accéder. Il suffit de réutiliser des mécanismes de confidentialité pour l'infrastructure de communication et pour les services.

2.5.2 Confidentialité et communication

À cet égard, la confidentialité implique qu'aucune information protégée n'est rendue compréhensible par quelqu'un d'autre que l'exploiteur prévu. Plusieurs modèles de communication qui supportent des mécanismes de chiffrement assurant la confidentialité ont été proposées. Par exemple, en mettant des mécanismes "*end-to-end encryption*" selon l'approche parcellaire, un message est chiffré par le service émetteur et il n'est déchiffré que par le service destinataire. Le contenu de message n'est pas exposé pendant son échange par le biais d'une chaîne de services d'interaction et de réseau de l'infrastructure de communication en commune. En mettant en œuvre des mécanismes "*link encryption*" selon l'approche décentralisée, un message est chiffré et déchiffré plusieurs fois durant son échange par le biais d'une chaîne de services d'interaction et de réseau de l'infrastructure de communication en commune. A chaque fois qu'il traverse un service de la chaîne de services d'interaction et de réseau, il est déchiffré et ensuite re-chiffré. La mise en œuvre des mécanismes assurant l'intégrité et la confidentialité sous forme d'un service omniprésent contribue à un canal sécurisé de communication par le biais d'une chaîne de services d'interaction et de réseau. Par exemple, la mise en œuvre de Kerberos [ker02] sous forme un service omniprésent fournit ces fonctionnalités.

2.5.2a Services d'interaction génériques ou dédiés à chaque modèle de coordination

Des mécanismes assurant la confidentialité sont mis en œuvre selon l'une des trois approches : parcellaire, décentralisée ou centralisée.

Communication orientée contrôle. Des travaux existants consistent à assurer la confidentialité des informations échangées (en tant que message ou évènement).
Il s'agit de mécanismes qui accompagnent des services qui communiquent des messages dans le cadre du plan de coordination. Ils sont souvent mis en œuvre selon l'approche parcellaire, par exemple les services d'interaction qui utilisent XML Encryption [xml02]. Ils assurent la confidentialité des messages échangés à travers une chaîne dynamique des services d'interaction et de réseau de l'infrastructure de communication. Par ailleurs, il s'agit de mécanismes de chiffrement partiel des messages échangés. Par exemple, les services d'interaction implantant le protocole SET (Secure Electronic Transaction [rfcb, set97]) permettent l'échange des données bancaires chiffrées dans le cadre des transactions entre le client (service navigateur), le vendeur (service Web) et la banque (service bancaire).

Communication orientée données. Des travaux existants consistent à assurer la confidentialité des informations sensibles partagées. Une telle protection est réalisée par un chiffrement des informations dans l'espace de données partagé (quelque soient des données localisées sur une machine ou sur plusieurs machines), ainsi qu'un contrôle d'accès rigoureux.

2.5.2b Services d'interaction et de réseau de l'infrastructure de communication

Il s'agit de mécanismes de chiffrement et de contrôle d'accès qui sont mis en œuvre selon des approches parcellaire ou décentralisée. Ces mécanismes accompagnent des services d'interaction

et de réseau ; ils assurent la confidentialité de tous les messages échangés par le biais de ces services. Ces messages sont à envoyer sans la clef publique de chiffrement (dans le cas d'utilisation du chiffrement symétrique tel que [sha04, fip01]) ou avec la clef publique (dans le cas d'utilisation du chiffrement asymétrique tel que [rfcc]). Par ailleurs, il s'agit de services de confidentialité sélective qui assure la confidentialité de certaines parties des messages échangés ou des données stockées selon les besoins confidentiels spécifiques tel que [ssl96]. Des services de confidentialité sont utilisés notamment à l'égard de communication, sur tous les aspects de la chaîne de services de communication.

2.5.3 Confidentialité et services à exploiter

À cet égard, l'idée est de bien contrôler l'intimité numérique des informations qui appartiennent en propre au service, ainsi que d'assurer leur confidentialité. Dans la plupart des cas, des contraintes de confidentialité du service sont reliées à la confidentialité durant la communication : il s'agit de contraintes spécifiant quels services de confidentialité seront utilisés pour la communication. En vue de la disponibilité du service, il s'agit de services de confidentialité non orientée communication (accompagnant des services qui communiquent des messages et qui assure l'intimité numérique des données à échanger). Leurs fonctions mettent en œuvre principalement des mécanismes de chiffrement. Par ailleurs, des mécanismes de protection supportés par la plate-forme de déploiement du service jouent un rôle très important. Des mécanismes assurant la confidentialité sont reliés à l'accessibilité et à la flexibilité d'un service, dans le sens où le service n'autorise pas des appels dont il ne peut pas déchiffrer leurs contenus. Ils sont reliés à l'inter-opérabilité : le service est interopérable aux exploiteurs qui utilisent les mêmes mécanismes assurant la confidentialité durant leur communication.

2.6 Non-répudiation[11]

D'une manière générale, la non-répudiation consiste à vérifier l'adéquation et la conformité de contrôle et d'exécution des fonctions d'un service à ses contraintes et ses propriétés pré-décrites, particulièrement celles de sécurité. Dans le contexte de l'exécution coordonnée, elle permet de prouver si un service a été maintenu dans l'état prévu pour des exploiteurs. Elle permet également de prouver l'apparition de certains évènements sur un service (accès, modification, etc.) ainsi que de prouver la responsabilité de ceux qui causent ces évènements. De plus, elle permet de prouver si aucune modification non autorisée n'a été faite au nom d'un exploiteur. Tout cela est faisable à condition que (i) les services fournissent des propriétés d'intégrité servant à vérifier leur intégrité ; (ii) les services fournissent des propriétés d'authentification servant à vérifier leur identité et l'origine des informations qui les concernent ; et (i) il existe suffisamment de preuves d'exécution pour des évènements à vérifier. Ces propriétés et preuves d'exécution sont considérées comme des propriétés de non-répudiation. Les mécanismes assurant la non-répudiation se basent donc sur les mécanismes d'authentification d'autorisation et d'intégrité. Les mécanismes de chiffrement asymétrique comme la signature ou la certification sont largement utilisés. Les mécanismes assurant la non-répudiation sont légèrement distingués selon leur fonctionnement (voir la figure 2.20).

[11] "A security service that provide protection against false denial of involvement in an association (especially a communication association that transfers data)." [rfca]

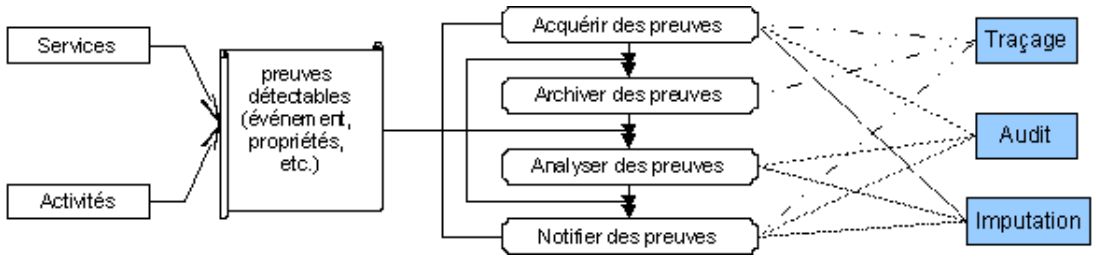


FIG. 2.20 – Principe de non-répudiation

Traçage Ces mécanismes archivent et rendent accessibles des preuves de différents aspects d'exécution d'un service pour reconstituer un historique des évènements sur un ou plusieurs aspects d'exécution. Les preuves d'exécution sont archivées par plusieurs services : ceux qui exécutent, ceux qui surveillent l'exécution, etc.

Par exemple, des techniques de journalisation de fonctions ou de messages fournies par des applications, par des systèmes d'exploitation, par des protocoles de communication, etc.

Audit Ces mécanismes collectent les preuves d'exécution des services, des exploiteurs, des activités en surveillant ces derniers ou en récupérant des preuves d'exécution archivées. Ils analysent ensuite ces preuves d'exécution afin de (i) détecter des violations de contraintes ou de propriétés d'un service ou (ii) empêcher des activités non autorisées causées par un exploiteur. Par exemple, des techniques ou outils de détection des intrusions, des mécanismes de déchiffrement d'une signature ou d'un *time-stamp* pour vérifier le contenu d'une preuve d'exécution, etc.

Imputation Ces mécanismes génèrent, diffusent et analysent des preuves d'exécution pour juger la répudiation les différends surgissant. Des preuves d'exécution sont générées et diffusées de manière franche (des preuves d'exécution sont générées et diffusée sans priorité pour tous les services qui font partie d'une exécution coordonnée) ou non franche (des preuves d'exécution sont générées et diffusées avec priorité pour certains services qui font partie d'une exécution coordonnée).

Deux approches ont été proposées pour la mise en œuvre des mécanismes d'analyse en se basant sur l'origine des preuves d'exécution collectées.

• **Sans tiers de confiance** Cette approche consiste en l'analyse des preuves d'exécution qui ne sont démontrées que par des services ayant un différend durant leur communication. La mise en œuvre des mécanismes de non-répudiation s'appuie sur deux cas généraux : (i) la non-répudiation de l'envoi (la preuve à l'expéditeur qu'un message ou un évènement a été envoyé) et (ii) la non-répudiation de la réception (la preuve au destinataire quant à l'origine d'information). Par exemple, le service implantant le protocole de non-répudiation à base de probabilité [MR99].

• **Avec tiers de confiance** Cette approche consiste en l'analyse des preuves d'exécution ; ces dernières sont démontrées soit par des services ayant un différend durant la communication, soit par des services les concernant (par exemple les services d'interaction utilisés, ou les autorités de certification qui certifient l'authenticité des services ayant un différend). Des mécanismes (par exemple des protocolesFNP [Zho01], CMP [ZOL05]) qui sont mis en œuvre selon cette approche permettent de plus de vérifier :

- la non-répudiation de soumission (la preuve à service d'interaction utilisé par l'expéditeur pour l'envoi d'un message ou d'un évènement) ;
- la non-répudiation de la livraison (la preuve à service d'interaction utilisé par l'expéditeur pour la réception d'un message ou d'un évènement par le destinataire) ; et éventuellement
- la non-répudiation de propriétés et contraintes de sécurité d'un service (la preuve aux services de sécurité sur l'attribution d'une propriété ou sur l'imposition d'une contrainte de sécurité aux services concernants).

2.6.1 Non-répudiation et plan de coordination

À l'égard de la coordination, la non-répudiation assure que
- les activités n'ont pas enfreint le plan (*trace*) ;
- les activités ayant subi une attaque, les informations ou services étant compromis à une menace effective sont ultérieurement détectés (*audit*) ;
- un service a effectué une activité de manière certaine (*imputation*).

2.6.1a En vue de l'expression sémantique de l'exécution coordonnée de services

La non-répudiation d'envoi, de réception, de soumission ou de livraison n'est pas très liée à l'expression de la sémantique de l'exécution coordonnée de services. Néanmoins, les critères suivants doivent être fournis lors de l'expression de la sémantique de l'exécution coordonnée.
- La non-répudiation d'ordonnancement : les preuves aux services fournissant des fonctions à coordonner pour l'ordre et la transition des activités du plan de coordination.
- La non-répudiation d'homologation : les preuves à tous les services utilisés (en dehors des services fournissant des fonctions à coordonner) dans le cadre du plan de coordination pour l'homologation des informations reliées aux activités.
- La non-répudiation de perception : les preuves à tous les services pour la perception des informations reliées aux activités.

Les modèles de coordination actuels ne permettent pas d'enrichir la sémantique d'exécution coordonnée avec l'aspect non-répudiation.

2.6.1b En vue de la gestion de l'exécution coordonnée de services.

Comme les autres aspects sécurité déjà considérés, les travaux actuels sur cet aspect sont orientés sur une architecture d'exécution du plan de coordination. La mise en œuvre des mécanismes existants de non-répudiation (selon l'une des approches centralisée, décentralisée ou parcellaire) permet de vérifier la non-répudiation d'envoi, de réception, de soumission et de livraison. Néanmoins, la non-répudiation d'ordonnancement, d'homologation ou de perception sur une activité exécutée dans le cadre du plan de coordination n'est pas facile à vérifier. Il y a des problèmes lors de la génération et de la collection des preuves d'exécution du plan de coordination pour juger la responsabilité des services qui participent à la coordination. Il manque une gestion générale de ces preuves ainsi que des problèmes d'intimité de service. Une telle vérification demande au préalable la capacité d'archiver et de rendre accessibles des informations afin de pouvoir tracer l'exécution des activités, ainsi que des informations nécessaires à une analyse ultérieure. Néanmoins, à cause de l'atomicité des services, un tel accès ne peut être obtenu dans la plupart des cas. Pour les solutions actuelles, les preuves d'exécution sont

collectées par des services de sécurité à différents niveaux (communication, application, réseau, système d'exploitation, etc.) d'une manière indépendante et non réutilisable.

2.6.2 Non-répudiation et communication

La non-répudiation d'envoi, de réception, de soumission, de livraison et éventuellement d'homologation est supportée à cet égard.

2.6.2a Services d'interaction dédiés ou génériques aux modèles de coordination

Communication orientée contrôle. Il s'agit principalement de l'imputation de l'envoi, de la réception et éventuellement d'homologation dans le cadre de protocoles sécurisés de communication qui supportent à la fois l'authentification, l'autorisation, l'intégrité et la confidentialité. Des mécanismes de traçage et d'audit sont aussi intégrés à de tels protocoles. Ces protocoles sont spécifiquement réservés à un modèle de coordination, donc les mécanismes d'imputation utilisés sont souvent non francs : ils permettent à un moteur de coordination de percevoir plus de preuves d'exécution que d'autres services. Les approches centralisée et décentralisée correspondent mieux à la non-répudiation sans tiers de confiance ; cela permet la communication directe entre les services qui utilisent un même service d'interaction. L'approche parcellaire correspond mieux à la non-répudiation avec tiers de confiance ; cela permet la mise en œuvre des mécanismes d'imputation francs.

Communication orientée données. Des mécanismes de non-répudiation sont établis autour des accès à l'espace de données partagé. Dans ce cas, ils reviennent aux mécanismes de non-répudiation à l'égard des services.

2.6.2b Services de l'infrastructure de communication

La non-répudiation pour des services de l'infrastructure de communication est supportée par des services implantant des protocoles sécurisés permettant l'authentification, l'autorisation, l'intégrité et la non-répudiation. Il s'agit de services sur des aspects transport et interconnexion.

Tandis que des mécanismes de traçage sont souvent mis en œuvre selon l'approche centralisée, des mécanismes d'imputation et d'audit sont mis en œuvre selon une de trois approches. L'approche parcellaire est la plus utilisée, en raison de la nécessité de l'inter-opérabilité entre des services implantant des mécanismes de non-répudiation qui sont associés à des services tiers de confiance ou services d'interaction dédiés. Dans le contexte de communication, il est important d'utiliser des mécanismes d'audit fournis par des services de non-répudiation : des activités effectuées sont toujours sous surveillance, et les résultats d'audit sont référencés comme la vérification rétrospective de sécurité. Des mécanismes de détection d'évènement ou détection d'intrusion sont aussi très importants : ils ont pour but d'observer des violations spécifiques de sécurité ou des évènements potentiellement dangereux, ou le nombre d'occurrences d'un évènement spécifique. Par ailleurs, afin d'assurer l'imputabilité, des mécanismes d'imputation supportés par des services de l'infrastructure de communication devraient être interopérables avec des mécanismes d'imputation des services qui les utilisent.

2.6.3 Non-répudiation et services à exploiter

Des preuves d'exécution d'un service sont collectées par des mécanismes de journalisation en divers niveaux : application, système d'exploitation, réseau, etc.

En vue de la disponibilité et l'accessibilité d'un service, il s'agit de services de non-répudiation qui mettent en œuvre des mécanismes d'authentification pour établir la responsabilité pour des activités sur un service. Une fois que la responsabilité est établie, il est facile de prouver l'imputation des activités. Des mécanismes de traçage, d'imputation et d'audit, en particulier des mécanismes de détection d'intrusion, sont très importants en vue de la disponibilité et la flexibilité d'un service. Ils permettent de tracer la source d'un problème de sécurité et de planifier la réparation. Néanmoins, la mise en œuvre inadéquate des mécanismes de non-répudiation peut dégrader la performance de services, en raison du volume des preuves collectées ainsi que de la fréquence d'audit.

2.6.4 Services de sécurité génériques

Il s'agit de services qui implantent des mécanismes de sécurité qui ne sont pas spécifiques ni à un aspect concret (authentification, autorisation, etc.), ni à un égard concret (plan de coordination, communication, service).

1. **Infrastructure de gestion de clefs** : Ce service fournit des fonctions de (i) gestion des clefs : la génération d'un couple unique de clefs, l'attribution des clefs aux services, l'archivage, etc. ; (ii) gestion des certificats numériques : création, signature, renouvellement des certificats, etc. ; (iii) diffusion des clefs publiques ; (iv) certification des clefs publiques (signature des certificats numériques). Par exemple, PKIX [rfc99a], NPKI [LC00], SPKI [spk99]. En pratique, une infrastructure de clefs publiques permet de gérer la distribution de clefs privées pour les différents utilisateurs (de même groupe ou de différentes locations) et le partage des clefs publiques entre eux. Il s'agit de fonctions permettant l'inscription, la certification et l'annuaire des clefs. Cette infrastructure est, quant à elle, mise en place par des *tiers de confiance*, qui produisent des certificats et les délivrent aux services.

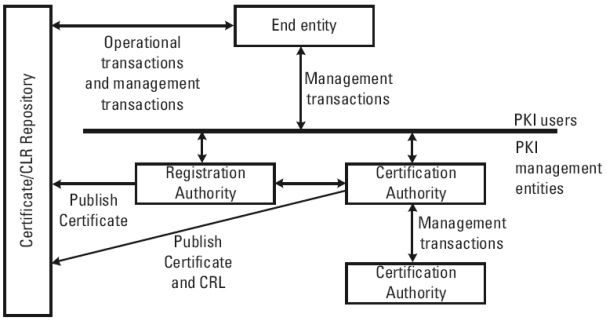


FIG. 2.21 – Infrastructure de gestion de clefs

2. **Schémas d'échange** : Il s'agit de services dont leurs fonctions mettent en œuvre différents schémas d'échange d'informations en utilisant des fonctions de chiffrement, de déchiffrement et de hachage. Un tel échange est supposé fournir l'identification pour les services qui communiquent. Il doit se produire avant la vérification de l'authentification, l'autorisation, l'intégrité, chiffrement ou non-répudiation.

3. **Tiers de confiance** : il s'agit de services de sécurité qui fournissent un certain niveau de confiance pour les services qui communiquent. Il s'agit de tiers de confiance "*inline*" (faisant partie de l'exécution coordonnée des services), "*online*" (à référencer par l'exécution coordonnée des services) ou "*offline*" (à référencer seulement en cas de différend).
   - **Services de certification** : Il s'agit de services permettant de gérer des patrons de confiance pour les propriétés, les documents et les logiciels.
   - **Services notaires** : Le fonctionnement principal d'un service notaire consiste à certifier la signature d'un identificateur, en se basant sur des preuves qu'il démontre et sur des preuves démontrées par d'autres services authentifiés qui le relient. En outre, un notaire produit des journaux indépendants de ses transactions. Chaque journal archive (i) le type de preuves qu'il a vérifié ; (ii) des signatures qui ont été indépendamment vérifiées par d'autres notaires précédents.

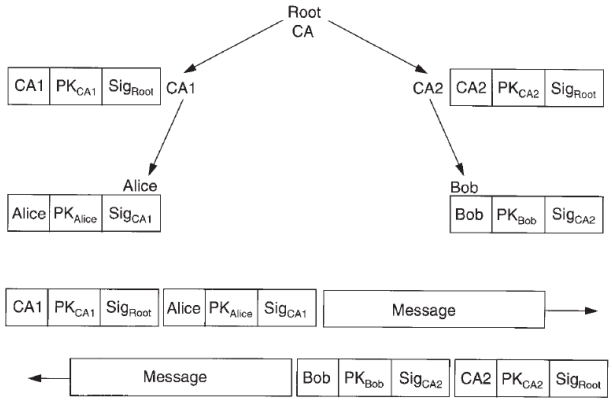


FIG. 2.22 – Services de certification et notaires hiérarchiques

Il s'agit de services permettant de chercher et de récupérer des clefs de chiffrement des informations échangées ; ces fonctions sont utilisées par des services de sécurité de tout aspect : authentification, autorisation, intégrité, confidentialité, non-répudiation.
Ces services fournissent également des fonctions servant à estampiller en temps réel (*time stamp*) des informations chiffrées sous forme de documents, ainsi que d'archiver des documents dépendants ; ces fonctions servent essentiellement aux services d'intégrité et de non-répudiation.
Par ailleurs, des services notaires permettent d'approuver des communications franches (*fair exchanges*). Il s'agit de fonctions permettant d'établir des contrats entre des services qui communiquent, ainsi que de récupérer des preuves de la soumission, de l'envoi et de la réception des messages. Ces fonctions servent essentiellement aux services de confidentialité et de non-répudiation.
   - **Services arbitraires** : Il s'agit de services permettant de donner la décision finale pour la légitimité d'un schéma d'authentification ou d'une preuve de non-répudiation.

4. **Services de dépannage** (*Security Recovery*) : Il s'agit de services permettant d'effectuer des dépannages de sécurité en vue de la gestion de l'exécution coordonnée de services.

5. **Services d'étiquette** (*Security Label*) : Il s'agit de services permettant d'étiqueter ou désigner des propriétés de sécurité pour les éléments constitutifs d'une architecture d'exécution coordonnée de services.

2.7 Conclusion

Sécuriser une application à base de services consiste à assurer la coordination sécurisée de ses activités. Cela implique la définition et la mise en œuvre des modèles de sécurité destinés aux aspects authentification, autorisation, intégrité, confidentialité et non-répudiation. Ces modèles visent à assurer un niveau de confiance en continu pour (i) les fonctions de services qui sont appelées dans le cadre des activités ; (ii) les données échangées entre services dans le cadre des activités et (iii) les activités d'un plan de coordination. Ils reposent sur les mécanismes de sécurité supportés par des services à coordonner et par l'infrastructure de communication.

Dans ce sens, la coordination sécurisée des activités d'une application à base de services doit refléter totalement des contraintes de sécurité typiques à satisfaire à l'égard de la spécification des besoins fonctionnels (logique applicative), ainsi qu'à l'égard de leur implantation.

Les modèles de sécurité pour la coordination devraient être à la fois orientés vers l'expression sémantique et orientés sur une architecture de l'exécution coordonnée de services. Néanmoins, en raison de l'inter-opérabilité, la disponibilité, l'accessibilité et la flexibilité de services utilisés (services à coordonner, services d'interaction et de réseau de l'infrastructure de communication), les modèles de sécurité actuels ne supportent que des mécanismes de sécurité qui sont cohérents et qui se ramènent à ceux qui sont supportés par et/ou appliqués sur les services utilisés, dans des contextes d'implantation et de déploiement concrets. Il s'agit donc des modèles qui ne sont qu'orientés sur une architecture d'exécution du plan de coordination. Les solutions ad-hoc proposées par ces modèles ne consistent qu'à sécuriser des composants spécifiques qui sont adressés à l'implantation de l'exécution coordonnée des services. Elles visent à spécifier et implanter (selon une de trois approches : parcellaire, centralisée et décentralisée) des fonctions, propriétés et contraintes de sécurité fournies / requises par ces composants (voir TAB. 2.4). Cette sécurisa-

		Authentification	Autorisation	Intégrité	Confidentialité	Non répudiation
Travaux orientés vers l'expression sémantique de l'exécution coordonnée de services : donner en plus de sémantique de sécurité lors de la définition du plan de coordination	Ordonnancement	o				o
	Déclenchement	o	o			o
	Interdépendance de données	o	o	o	o	o
Travaux orientés vers l'architecture de l'exécution coordonnée de services : sécuriser le moteur d'exécution du plan de coordination	Version sécurisée du moteur d'exécution du plan de coordination	o	x	o	o	x
	Configuration sécurisée du moteur d'exécution du plan de coordination	x	o	x	x	o
	Services sécurité dédiés / génériques, réutilisables par le moteur d'exécution du plan de coordination	x	x	x	x	x
Travaux orientés vers une coordination sécurisée, qui reflètent totalement les différents aspects de l'exécution coordonnée de services sur une infrastructure d'exécution de l'application à base de services		o	o	o	o	o

x : travaux existants o : travaux manquants case vide : non applicable

TAB. 2.4 – Travaux sur la coordination sécurisée

tion, quant à elle, se ramène à l'implantation des solutions de sécurité à l'égard des services et à l'égard de la communication. De cette façon, il n'y a aucun lien sémantique entre les besoins de coordination et les besoins de sécurité qui peut être considéré lors de la définition de l'exécution coordonnée de services (voir TAB. 2.5). Par conséquent, elle ne peut pas être adaptée en fonction

des besoins fonctionnels spécifiques d'applications à base de services.

			Mise en relation avec la définition de l'exécution coordonnée														Mise en relation avec l'architecture de l'exécution coordonnée				
			Ordonnancement					Déclenchement					Interdépendance de données									
			Authentification	Autorisation	Intégrité	Confidentialité	Non répudiation	Authentification	Autorisation	Intégrité	Confidentialité	Non répudiation	Authentification	Autorisation	Intégrité	Confidentialité	Non répudiation	Authentification	Autorisation	Intégrité	Confidentialité	Non répudiation
Service	Mise en relation avec la logique applicative de service		o	o				o		o		o						o	x	x	x	
	Mise en relation avec l'implantation de service		o	o				o		o		o			o		o	x	x	x		x
	Disponibilité														o		o		x	x		x
	Accessibilité		o	o	o		o					o						x	x	x		x
	Communication							o	o	o		o			o		o	x	x	x	x	x
	Flexibilité				o		o	o		o		o			o		o	x	x	x		
Infrastructure de communication	Services d'interaction	services dédiés (découverte, conversation, description, etc.)												o	o		o	x	x	x	x	x
		transfert des infos																				
		transformations des infos																			x	
		établissement et gestion des communications																			x	
	Services de réseau	transport												o	o		o	x	x	x	x	x
		interconnexion entre réseaux												o	o		o	x	x	x	x	x
		liaison de data grammes																				
		raccordement																				

x : travaux existants o : travaux manquants case vide : non applicable

TAB. 2.5 – Travaux sur la sécurité dans les applications à base de services

Chapitre 3

Le modèle de coordination sécurisée MEO

Ce chapitre présente MEO, un modèle de coordination sécurisée. Dans ce modèle, une application à base de services est vue comme un plan de coordination sécurisée d'activités. Une activité correspond à un appel à une fonction d'un service.
Du point de vue du plan de coordination sécurisée, une activité :
- *appartient à un plan de coordination sécurisée ;*
- *possède des propriétés nécessaires pour l'exécution de ce plan ;*
- *a conscience des services servant à son exécution dans ce plan ;*
- *est conditionnée par des contraintes de ce plan qui sont à satisfaire avant et après l'appel ;*
- *fournit des preuves d'exécution qui sont archivées dans des journaux de coordination ; ces preuves servent à l'exécution des autres activités du plan ainsi qu'aux analyses ultérieurs.*

Le plan de coordination sécurisée précise les instructions à réaliser sur les activités ; ces instructions concernent :
- *les aspects fonctionnels de coordination, décrites par des contraintes de coordination ;*
- *les aspects non-fonctionnels de sécurité, décrites par des contraintes de sécurité.*

Un tel plan est donc décrit sous forme d'un ensemble d'activités dont les contraintes devant être satisfaites sont définies par les formules bien formées, correctement associées, cohérentes et évaluables.
Le modèle MEO offre les concepts nécessaire à la description d'un plan (voir Fig. *3.1) :*
- *les* ***activité****s à exécuter avec certaines* ***propriété****s dans le cadre de la coordination sécurisée ;*
- *les* ***contrainte****s associées aux activités régissant la coordination sécurisée ; elles sont définies à partir de* ***prédicat****s*
- *les* ***journa****ux* ***de coordination*** *à base de* ***preuves d'exécution*** *des activités.*

Ce chapitre est organisé de manière suivante. La section 3.1 décrit les notions pré-requises du modèle MEO : type, domaine, opération et fonction. La section 3.2 décrit les concepts de base : propriété, service, contrainte, activité, journal de coordination. La section 3.3 détaille les propriétés et les prédicats de base pour la description des aspects relatifs à la coordination et à la sécurité ; ceux-ci servent à définir les contraintes du plan de coordination sécurisée. La section 3.4 montre comment les concepts du modèle sont utilisés. Elle décrit les règles d'association entre contraintes et activités du point de vue de leurs temps d'évaluation et un exemple de définition de plan de coordination sécurisée des activités.

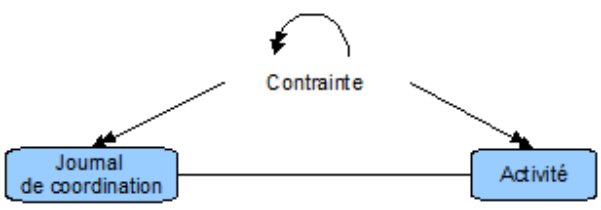


Fig. 3.1 – MEO - modèle de coordination sécurisée

3.1 Pré-requis

Les notions suivantes sont pré-requises pour la définition des concepts du modèle MEO.

3.1.1 Domaine

Un domaine Db est un ensemble de valeurs distinctes. Il contient aussi des valeurs "NULL"[1] :
- la non-information $\phi \in$ Db .
- une information inconnue à l'instance ? $\in$ Db.

Db :− v dénote la valeur v de Db. Un domaine qui se compose de valeurs non décomposables est un domaine atomique.

Nous considérons les domaines atomiques : Boolean, Char, String, Integer, Float, File, Message et Time.

Exemple. Integer :− 100 est une valeur du domaine atomique Integer.

$\mathcal{D}$ dénote l'union des domaines. Il inclut la valeur δ.

3.1.2 Type

Le domaine ∇ des types est défini par les règles suivantes :

1. Un domaine atomique Db est un type de base

$$\mathsf{Db} :- \mathsf{Db} \in \nabla \tag{3.1.1}$$

2. Un type construit T est défini de manière récursive :

$$\mathsf{T} :- \mathsf{tuple}(a_1 : \mathsf{T}_1, a_2 : \mathsf{T}_2, .., a_n : \mathsf{T}_n) \in \nabla \;\; \forall \mathsf{i}, \mathsf{j} \in [1..\mathsf{n}], \mathsf{i} \neq \mathsf{j}, a_i \neq a_j \tag{3.1.2}$$

$$\mathsf{T} :- \mathsf{choice}(\mathsf{T}_1, \mathsf{T}_2, .., \mathsf{T}_\mathsf{n}) \in \nabla \tag{3.1.3}$$

$$\mathsf{T} :- \mathsf{list}(\mathsf{T}_1) \in \nabla \tag{3.1.4}$$

$$\mathsf{T} :- \mathsf{set}(\mathsf{T}_1) \in \nabla \tag{3.1.5}$$

$$\mathsf{T} :- \mathsf{ref}(\mathsf{T}_1) \in \nabla \tag{3.1.6}$$

où T_1, T_2,.., $\mathsf{T}_\mathsf{n} \in \nabla$ sont des types et a_1, a_2,.., $a_n \in$ String sont des noms distincts.

3.1.2a Nom et domaine de types

Les fonctions $\mathsf{name} : \nabla \to \mathsf{String}$ et $\mathsf{dom} : \nabla \to \mathcal{D}$ permettent de retrouver le nom et le domaine d'un type T de ∇ [2] :
- Si T est un type de base de la forme Db :− Db $\in \nabla$ alors :

  name(Db :− Db) = Db ;

  dom(Db) = Db.

  Db dénote à la fois le domaine (Db $\subset \mathcal{D}$) et le type (Db $\in \nabla$).

  Exemple. name(Integer :− Integer) = Integer : Le nom du domaine (Integer) dénote également le nom du type.

[1] Une information non existante, dénotée δ, est une valeur "NULL" qui n'appartient à aucun domaine.

[2] Notons que :
$\forall \mathsf{T} \in \nabla$ dom(T) $\subset \mathcal{D}$: le domaine définissant un type T $\in \nabla$ est un domaine de $\mathcal{D}$.
$\forall \mathsf{T} \in \nabla$? $\in$ dom(T) : une valeur inconnue à l'instance est de n'importe quel type T $\in \nabla$.

$\mathsf{dom(Integer)} = \mathsf{Integer}$: le domaine définissant le type Integer est le domaine atomique Integer.

- Si $\mathsf{T} :- \mathsf{tuple}(a_1 : \mathsf{T}_1, a_2 : \mathsf{T}_2, .., a_n : \mathsf{T}_n) \in \nabla$ alors :

  $\mathsf{name}(\mathsf{T} :- \mathsf{tuple}(a_1 : \mathsf{T}_1, a_2 : \mathsf{T}_2, .., a_n : \mathsf{T}_n)) = \mathsf{T}$;

  $\mathsf{dom}(\mathsf{T}) = \{\mathsf{T} :- \mathsf{tuple}(a_1 : v_1, a_2 : v_2, .., a_n : v_n) \mid \mathsf{n} \geq 1, \forall \mathsf{i} \in [1..\mathsf{n}], v_i \in \mathsf{dom}(\mathsf{T_i})\}$ est un domaine de valeurs n-uplets, où chaque valeur $\mathsf{T} :- \mathsf{tuple}(a_1 : v_1, a_2 : v_2, .., a_n : v_n)$ est une agrégation des valeurs $v_1, v_2, .., v_n$ (resp. de types T_1, T_2,.., T_n).

- Si $\mathsf{T} :- \mathsf{choice}(\mathsf{T}_1, \mathsf{T}_2, .., \mathsf{T}_n) \in \nabla$ alors :

  $\mathsf{name}(\mathsf{T} :- \mathsf{choice}(\mathsf{T}_1, \mathsf{T}_2, .., \mathsf{T}_n)) = \mathsf{T}$;

  $\mathsf{dom}(\mathsf{T}) = \{\mathsf{T} :- v \mid v \in \mathsf{dom}(\mathsf{T_i}), \mathsf{i} \in [1..\mathsf{n}]\}$ est un domaine de valeurs alternatives, où chaque valeur $\mathsf{T} :- v$ est choisie parmi des valeurs typées $\mathsf{T}_1 :- v_1$, $\mathsf{T}_2 :- v_2$, .., $\mathsf{T}_n :- v_n$.

- Si $\mathsf{T} :- \mathsf{list}(\mathsf{T}_1) \in \nabla$ alors :

  $\mathsf{name}(\mathsf{T} :- \mathsf{list}(\mathsf{T}_1)) = \mathsf{T}$;

  $\mathsf{dom}(\mathsf{T}) = \{\mathsf{T} :- \mathsf{list}(v_1, v_2, .., v_n) \mid \mathsf{n} \geq 1, \forall \mathsf{i} \in [1..\mathsf{n}], v_i \in \mathsf{dom}(\mathsf{T}_1)\}$ est un domaine de listes, où chaque valeur $\mathsf{T} :- \mathsf{list}(v_1, v_2, .., v_n)$ est une liste des valeurs $v_1, v_2, .., v_n$ (de type T_1).

- Si $\mathsf{T} :- \mathsf{set}(\mathsf{T}_1) \in \nabla$ alors :

  $\mathsf{name}(\mathsf{T} :- \mathsf{set}(\mathsf{T}_1)) = \mathsf{T}$;

  $\mathsf{dom}(\mathsf{T}) = \{\mathsf{T} :- \mathsf{set}(v_1, v_2, .., v_n) \mid \mathsf{n} \geq 1, \forall \mathsf{i} \in [1..\mathsf{n}], v_i \in \mathsf{dom}(\mathsf{T}_1)\}$ est un domaine d'ensembles, où chaque valeur $\mathsf{T} :- \mathsf{set}(v_1, v_2, .., v_n)$ est un ensemble de valeurs distinctes $v_1, v_2, .., v_n$ (de type T_1).

- Si $\mathsf{T} :- \mathsf{ref}(\mathsf{T}_1) \in \nabla$ alors :

  $\mathsf{name}(\mathsf{T} :- \mathsf{ref}(\mathsf{T}_1)) = \mathsf{T}$;

  $\mathsf{dom}(\mathsf{T}) = \{\mathsf{T} :- \mathsf{ref}(v_1) \mid v_1 \in \mathsf{dom}(\mathsf{T}_1)\}$ est un domaine de références, où chaque valeur $T :- \mathsf{ref}(v_1)$ est une référence à une valeur v_1 de type T_1.

3.1.2b Intention et extension de types

T dénote à la fois :

- son extension, c.-à-d. le domaine de ses valeurs : $\{\mathsf{v} \mid \mathsf{v} \in \mathsf{dom}(\mathsf{T})\}$
- son intention, c.-à-d. son nom :
  - Le prédicat $\mathsf{T}(\mathsf{v})$ indiquant si une valeur v appartient au domaine définissant le type T.
  - Les axiomes définissant ses opérations ; ils permettent de valider l'utilisation de ces opérations.

3.1.3 Opération

Les opérations d'un type T définissent le comportement de ses valeurs. Une opération possède un nom (*operationName*), une description de ses paramètres d'entrée (*input*) et une description de son résultat de sortie (*output*). Elle est décrite comme une instance du (meta) type :

$$\begin{aligned}\mathsf{Operation} :- \mathsf{tuple}(&\\ &\mathit{operationName} : \mathsf{String},\\ &\mathit{input} : \mathsf{list}(\mathsf{tuple}(\mathit{paramName} : \mathsf{String}, \mathit{paramType} : \mathsf{String})), \qquad \in \nabla \quad (3.1.7)\\ &\mathit{output} : \mathsf{tuple}(\mathit{resultName} : \mathsf{String}, \mathit{resultType} : \mathsf{String})\\ )&\end{aligned}$$

Le type ($\in \nabla$) du premier paramètre d'une opération est le type auquel est associée cette opération. Ce paramètre est le premier élément de la liste *input*, c.-à-d. une valeur tuple(*paramName* : ?, *paramType* : T). L'opération est alors appelée l'opération du type T (ou l'opération définie sur le domaine dom(T))[3].

Exemple. Soit le domaine Account $\subset$ String qui caractérise les noms des utilisateurs d'un service, et le domaine Password $\subset$ String qui caractérise des mots de passe chiffrés.

$$\begin{aligned}\mathsf{Operation} :- \mathsf{tuple}(&\\ &\mathit{operationName} : \mathit{login},\\ &\mathit{input} : \mathsf{list}(\mathsf{tuple}(\mathit{paramName} : \mathit{username}, \mathit{paramType} : \mathit{Account}),\\ &\qquad \mathsf{tuple}(\mathit{paramName} : \mathit{pwd}, \mathit{paramType} : \mathit{Password})),\\ &\mathit{output} : \mathsf{tuple}(\mathit{resultName} : \mathit{result}, \mathit{resultType} : \mathit{Boolean})\\ )&\end{aligned}$$

est une instance du type Operation qui définit la signature de l'opération login du type Account. Nous l'écrivons plus simplement :

$$\mathsf{login} : (\mathit{username} : \mathsf{Account}) \times (\mathit{pwd} : \mathsf{Password}) \rightarrow (\mathit{result} : \mathsf{Boolean})$$

Nous supposons l'existence :

- des opérations de comparaison $>, <, =, \neq, \leq, \geq$, définies sur les domaines de base et les domaines de référence ;
- des opérations $\in, \notin, \cup, \cap, \subset, \supset, \subseteq, \supseteq$, définies sur les domaines n-uplet, ensemble et liste.

3.1.4 Fonction

Le nom d'une opération d'un type T est utilisé comme nom de fonction. Le nombre de paramètres de la fonction définit son arité.

$\mathcal{F}_{\mathsf{T}}$ dénote le domaine des fonctions du type T, c.-à-d. les fonctions dont le premier paramètre est de type T.

$\mathcal{F}$ dénote l'union des domaines des fonctions de tous les types de ∇.

Soit la fonction n-aire $\mathsf{f} \in \mathcal{F}_{\mathsf{T}}$ décrite par :

[3] Une opération sans paramètre est de n'importe quel type.

$$\begin{array}{l}\mathsf{Operation} :- \mathsf{tuple}(\\ \quad \mathit{operationName} : f,\\ \quad \mathit{input} : \mathsf{list}(\mathsf{tuple}(\mathit{paramName} : param_1, \mathit{paramType} : T),\\ \quad\quad \mathsf{tuple}(\mathit{paramName} : param_2, \mathit{paramType} : T_2)), ..,\\ \quad\quad \mathsf{tuple}(\mathit{paramName} : param_n, \mathit{paramType} : T_n)),\\ \quad \mathit{output} : \mathsf{tuple}(\mathit{resultName} : result, \mathit{resultType} : T_m)\\ )\end{array}$$

Lorsque nous l'appliquons à $v \in \mathsf{dom}(\mathsf{T}), v_2 \in \mathsf{dom}(\mathsf{T_2}), .., v_n \in \mathsf{dom}(\mathsf{T_n})$, nous écrivons $f(\mathsf{T} :- v, \mathsf{T_2} :- v_2, .., \mathsf{T_n} :- v_n)$ ou plus simplement $f(v, v_2, .., v_n)$. Cela dénote une instance de l'opération f définie pour une valeur $\mathsf{T} :- v$.

Exemple. login(guest, foo) est une instance de l'opération login.

3.2 Concepts de base

"Propriété", "service", "contrainte", "activité" et "journal de coordination" sont les concepts de base du modèle MEO pour définir une coordination sécurisée.

3.2.1 Propriété

Une propriété est une information intrinsèque appartenant à un service ou à une activité. Elle porte un nom et possède une valeur typée.

Toute propriété est une instance du (meta) type :

$$\mathsf{Property} :- \mathsf{tuple}(\mathit{propertyName} : \mathsf{String}, \mathit{propertyValue} : \mathsf{T}) \qquad \in \nabla \qquad \text{où } \mathsf{T} \in \nabla \quad (3.2.1)$$

Une instance d'une propriété est de la forme

$$\mathsf{Property} :- \mathsf{tuple}(\mathit{propertyName} : \mathsf{p}, \mathit{propertyValue} : (\mathsf{T} :- \mathsf{v}))$$

où *p* est le nom de la propriété et $\mathsf{T} :- v$ est une valeur (de type T[4]) pour cette propriété. Nous l'écrivons plus simplement :

$$p : \mathsf{v}$$

Le nom d'une propriété est utilisé comme une étiquette qui est attachée aux informations définissant la coordination sécurisée afin de faciliter leur exploitation.

dom(Property) dénote le domaine des propriétés, $\mathsf{dom}(\mathsf{Property}) \subset \mathcal{D}$.

Exemple. *activityId* : String définit la propriété de nom *activityId* et de type String. Cette propriété caractérise les noms des activités d'un plan de coordination sécurisée.
Pour attribuer le nom A3a à une activité, nous écrivons *activityId* : A3a
Cela indique donc que la propriété de nom *activityId* de cette activité a pour valeur A3a (de type String). Autrement dit, la valeur A3a est étiquetée comme *activityId*.

[4] T est donc le type associé au nom de la propriété. Le type associé effectif d'une propriété est déterminé de manière récursive, dans le cas où une propriété est définie à partir d'une autre propriété.

approval : (*activityId* : A3a) définit la propriété de nom *approval* et la valeur *activityId* : A3a (de type Property) pour cette propriété. Autrement dit, la valeur A3a (de type String) est étiquetée à la fois comme *activityId* et *approval*.

Exemple. Property :– tuple(*propertyName* : authenSubjProof, *propertyValue* : Certificate) définit une propriété d'authentification de type Certificate.

Exemple.

```
Property :– tuple(
    propertyName : communication,
    propertyValue : tuple(
        proxies : set(Service),
        bindings : set(tuple(protocol : choice(Service, Operation),
                datapackets : tuple(sentmsg : MsgFormat, recvmsg : MsgFormat))))
)
```

définit une propriété de communication d'un service ou d'une activité. Cette propriété caractérise les chemins d'accès, les protocoles de communication et les formats de données échangées supportés. Le type n-uplet associé au nom de cette propriété est construit à partir des types Service, Operation et MsgFormat.
Une instance de la propriété *communication* ∈ dom(Property) est définie par :

```
communication : tuple(
    proxies : set(ref(Agence), ref(http : //www – lsr.imag.fr/HADAS : 8080)),
    bindings : set(
        tuple(protocol : ref(SOAPMsgSrv), datapackets : list(sentmsg : ref(SOAPM1), recvmsg : ref(SOAPM2))),
        tuple(protocol : ref(JavaMessageService), datapackets : list(sentmsg : ref(JMsg1), recvmsg : ref(JMsg2))))
)
```

Autrement dit, cette valeur n-uplet est étiquetée comme *communication*.

3.2.2 Service

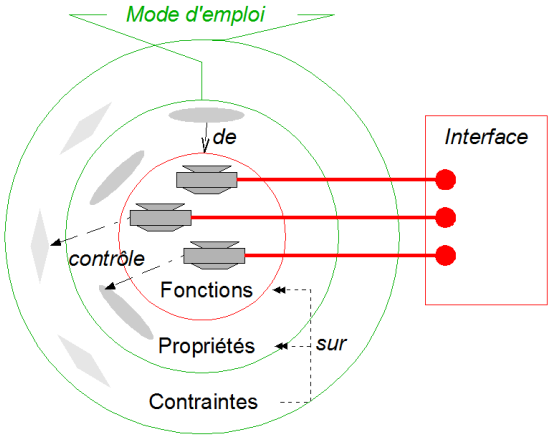


FIG. 3.2 – Service

Un service est caractérisé par un nom (*serviceName*), un ensemble de fonctions (*signatures*), un ensemble de propriétés du service (*properties*) et un ensemble de contraintes sur les fonctions et les propriétés du service (*constraints*).

Le type Service $\in \nabla$ caractérise des services. Il est défini par :

$$\begin{array}{l}\mathsf{Service} :\!- \mathsf{tuple}(\\ \quad \mathit{serviceName} : \mathsf{String},\\ \quad \mathit{signatures} : \mathsf{set}(\mathsf{Operation}),\\ \quad \mathit{properties} : \mathsf{set}(\mathsf{Property}),\\ \quad \mathit{constraints} : \mathsf{set}(\mathsf{Formula})\\ )\end{array} \in \nabla \qquad (3.2.2)$$

Les instances du type Service sont construites en donnant les valeurs à la structure tuple : nom, signatures des fonctions, propriétés et contraintes.

La figure 3.2 illustre un service qui exporte ses fonctions à travers son interface. Il exporte également les propriétés et les contraintes de différents aspects fonctionnels et non-fonctionnels de son exécution dans le mode d'emploi du service.

Exemple. Le service Agence fournit des fonctions permettant l'interaction et la gestion des clients qui s'intéressent à la réservation des places ; la recherche des places disponibles ainsi que la pré-réservation des places choisies sur des bases de données de vols de différents opérateurs. Le mode d'emploi de ce service spécifie les contraintes et les propriétés des aspects fonctionnels et non-fonctionnels de son exécution : la preuve de contrôle d'accès, la visibilité de fonctions de ce service, l'ordre dans lequel les fonctions du service doivent être appelées ainsi que le service d'interaction utilisé lors de la communication, etc. Un tel service est partiellement défini comme suit :

```
Service :− tuple(
 serviceName : Agence,
 signatures : set(
  login : (username : Account) × (pwd : Password) → (result : Boolean),
  selectFlight : (fullflightdescs : list(File)) → (chosenflightdesc : File),
  ?
 ),
 properties : set(
  authorSubjProof : ref(certificate1),
  communication : tuple(
   proxies : set(ref(Agence), ref(http : //www − lsr.imag.fr/HADAS : 8080)),
   bindings : set(
    tuple(protocol : ref(SOAPMessageService), datapackets : list(sentmsg : SOAPMessage, recvmsg : SOAPMessage)),
    tuple(protocol : ref(JavaMessageService), datapackets : list(sentmsg : JavaMessage, recvmsg : JavaMessage)))
  ),
  ?
 ),
 constraints : set(
   public(set(ref(login), ref(displayResult), ref(selectFlight))),
   after(ref(selectFlight), ref(login)),
   required(Certificate(), Service(authenObjProof ())),
  ?
 )
)
```

Les instances du type Service sont également construites en faisant référence à un service :

on est alors en mesure de déterminer les valeurs de la structure tuple par une valeur de type ref().

Exemple. Service :– ref(Agence) définit un service par une valeur qui fait référence au nom d'un service existant.

3.2.3 Contrainte

Une contrainte est une formule bien formée définie à partir de termes et de prédicats.

3.2.3a Terme

Un terme est défini de manière récursive par les règles suivantes :
- une constante c (qui est l'élément d'un domaine de $\mathcal{D}$) est un terme ;
- une variable v (qui est définie sur un domaine de $\mathcal{D}$) est un terme ;
- si $\mathtt{f}$ est une fonction n-aire ($\mathtt{f} \in \mathcal{F}$) et $\mathtt{t_1}, \mathtt{t_2}, .., \mathtt{t_n}$ sont des termes alors $\mathtt{f}(\mathtt{t_1}, \mathtt{t_2}, .., \mathtt{t_n})$ est un terme.

3.2.3b Prédicat

Un prédicat est une formule à base de termes. Le domaine des prédicats $\mathcal{P}$ ($\mathcal{P} \subset \mathcal{D}$) est défini par les règles suivantes :

1. $\forall \mathsf{T} \in \nabla$ alors $\mathtt{T}() \in \mathcal{P}$

   Exemple. Integer() est le prédicat qui vérifie si une valeur donnée est de type Integer (si elle appartient au domaine Integer) :
   Integer(100) est évalué à $VRAI$; Integer(A3a) est évalué à $FAUX$.

2. Une propriété est un prédicat : $\forall \mathsf{P} \in \mathsf{dom}(\mathsf{Property})$ alors $P() \in \mathcal{P}$

   Exemple. *activityId*() est le prédicat qui vérifie si une valeur donnée est de type String et si elle caractérise le nom d'une activité d'un plan de coordination sécurisée : activityId(A3a) est évalué à $VRAI$; activityId(100) est évalué à $FAUX$.

3. Une fonction n-aire qui retourne une valeur de type Boolean est un prédicat :
   $\forall \mathsf{f} : (\mathit{param_1} : \mathsf{T}) \times (\mathit{param_2} : \mathsf{T_1}) \times .. \times (\mathit{param_n} : \mathsf{T_n}) \rightarrow (\mathit{result} : \mathsf{Boolean}) \in \mathsf{dom}(\mathsf{Operation})$
   alors $f(T(), T_1(), .., T_n()) \in \mathcal{P}$

   Exemple. login(Account(), Password()) est un prédicat $\in \mathcal{P}$;
   login(Account(guest), Password(foo)) est évalué à $VRAI$;
   login(Account(100), Password(foo)) est évalué à $FAUX$.

4. Nous considérons également des prédicats qui sont utiles à la définition des contraintes de différents aspects d'un plan de coordination sécurisée[5].
   Soit $\mathtt{p} \in \mathtt{String}$ un terme constant représentant le nom du prédicat à définir, $\mathtt{P_1}(), .., \mathtt{P_n}() \in \mathcal{P}$ des prédicats, alors $\mathtt{p}(\mathtt{P_1}(), .., \mathtt{P_n}())$ est un prédicat.

   Exemple. before(Activity(), Activity()) est un prédicat pré-défini qui permet de vérifier si une activité se produit avant une autre.
   before(Activity(*activityId*(A3)), Activity(*activityId*(A4))) est évalué à $VRAI$ si l'activité de nom A3 se produit avant l'activité de nom A4 ;
   before(*activityId*(A3), *activityId*(A3)) est évalué à $FAUX$.

[5] De tels prédicats sont décrits en détails dans la section 3.3.

Nous choisissons les prédicats de $\mathcal{P}$ comme la granularité des formules. Un prédicat est donc une formule atomique.

3.2.3c Formule bien formée

Une formule bien formée est décrite à l'aide d'un alphabet de symboles qui comporte :
- l'ensemble des constantes, éléments des domaines de $\mathcal{D}$ [6] ;
- un ensemble de variables v_i définies sur les domaines de $\mathcal{D}$ [6] ;
- l'ensemble des fonctions avec arité ($\mathcal{F}$) [6] ;
- l'ensemble des prédicats avec arité ($\mathcal{P}$) ;
- les connecteurs logiques $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, les quantificateurs $\ni, \exists, \forall$ et les symboles de ponctuation ().

Le domaine Formula (Formula $\subset \mathcal{D}$) des formules bien formées est défini de manière récursive par des règles suivantes :

1. les termes *VRAI* et *FAUX* sont des formules bien formées ;
2. une formule atomique (c.-à-d. un prédicat) est une formule bien formée ;
3. si `f` est une formule bien formée alors $\neg$`f` est une formule bien formée ;
4. si $\mathtt{f}_1$ et $\mathtt{f}_2$ sont des formules bien formées alors $\mathtt{f}_1 \wedge \mathtt{f}_2$, $\mathtt{f}_1 \vee \mathtt{f}_2$, $\mathtt{f}_1 \rightarrow \mathtt{f}_2$, $\mathtt{f}_1 \leftrightarrow \mathtt{f}_2$ sont des formules bien formées ;
5. si `f` est une formule bien formée alors $\exists$`f`, $\forall$`f`, $\ni$ `f` sont des formules bien formées ;
6. si `f` est une formule bien formée alors (`f`) est une formule bien formée.

Toute formule bien formée est une instance du type Formula qui est défini par le domaine Formula $\subset \mathcal{D}$:

$$\mathsf{Formula} :- \mathsf{Formula} \in \nabla \tag{3.2.3}$$

Autrement dit, toute contrainte est une instance du type Formula.

La figure 3.3 illustre trois familles de contraintes régissant la coordination sécurisée : contraintes spécialisées (C.S.), contraintes locales (C.L.) et contraintes globales (C.G.). Elles sont détaillées dans les sections suivantes.

3.2.3d Contrainte spécialisée

Une contrainte spécialisée est une formule bien formée. Cette dernière est construite de manière non récursive (conformément aux règles de définition des formules bien formées, voir la section 3.2.3c) à partir (i) des prédicats qui sont pré-définis pour un aspect spécifique d'un plan de coordination sécurisée, (ii) des connecteurs logiques et (iii) des quantificateurs.

Une contrainte spécialisée est associée à une activité spécifique du plan de coordination sécurisée d'activités. Dans la figure 3.3, les losanges nommés C.S. illustrent les contraintes spécialisées d'un plan. Une contrainte spécialisée se rattache à l'appel (à exécuter dans le cadre de cette activité) sans en constituer l'essentiel. Elle est à évaluer dans le cadre de l'exécution de cette activité (c.-à-d. à satisfaire avant ou après l'exécution de l'appel correspondant).

Du point de vue d'une activité, une contrainte spécialisée désigne une qualité de l'exécution de cette activité ou une qualité d'un service servant à cette exécution.

[6] Il s'agit des symboles décrivant des termes.

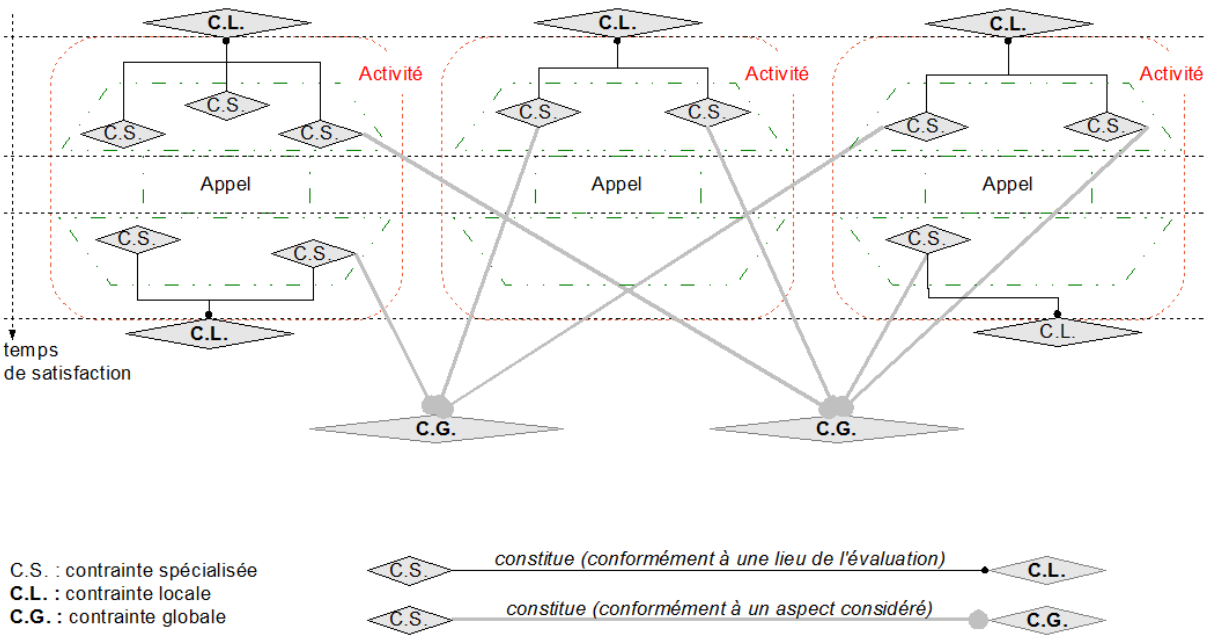


FIG. 3.3 – Contraintes régissant la coordination sécurisée

Du point de vue d'un plan de coordination sécurisée d'activités, une contrainte spécialisée désigne une relation entre une activité et (i) une autre activité de même plan ; et/ou (ii) un service servant à son exécution.

La qualité ou la relation désignée par une contrainte spécialisée ne concerne qu'un aspect spécifique d'un plan de coordination sécurisée.

Exemple. $\texttt{before}(\texttt{Activity}(\textit{activityId}(\mathsf{A3a})), \texttt{Activity}(\textit{activityId}(\mathsf{A4})))$ (1) est un prédicat spécifiant que l'activité de nom A3a doit se produire avant l'activité de nom A4. Il est à évaluer dans le cadre de l'activité de nom A3a (avant l'exécution de l'appel correspondant), alors cette formule bien formée définit une contrainte spécialisée.
De même, $\texttt{before}(\texttt{Activity}(\textit{activityId}(\mathsf{A4})), \texttt{Activity}(\textit{activityId}(\mathsf{A5}))) \in \mathcal{P}$ (2) est un prédicat spécifiant que l'activité de nom A4 doit se produire avant l'activité de nom A5. Il est à évaluer dans le cadre de l'activité de nom A4 (avant l'exécution de l'appel correspondant), alors cette formule bien formée définit une contrainte spécialisée.
La disjonction de deux prédicats (1) et (2) est aussi une formule bien formée ; elle spécifie que les activités A3a, A4 et A5 se produisent consécutivement. Cette formule définit donc une contrainte. Néanmoins, elle ne définit pas une contrainte spécialisée car les prédicats la constituant sont à évaluer dans le cadre de deux activités différentes.

3.2.3e Contrainte locale

Lorsque les contraintes spécialisées sont associées à la même activité d'un plan de coordination sécurisée, elles doivent être satisfaites conjointement avant ou après l'exécution de l'appel dans le cadre de cette activité.

Une contrainte locale est une formule bien formée qui est construite de manière non récursive (conformément aux règles de définition des formules bien formées, voir la section 3.2.3c) à partir de contraintes spécialisées et de connecteurs logiques. Par défaut, cette formule correspond soit à la conjonction de toutes contraintes spécialisées devant être satisfaites avant l'exécution de

l'appel dans le cadre d'une activité, soit à la conjonction de toutes contraintes spécialisées devant être satisfaites après l'exécution de l'appel dans le cadre d'une activité (voir la figure 3.3).

Du point de vue d'une activité d'un plan de coordination sécurisée, une contrainte locale régit l'exécution de cette activité en fonction de la satisfaction des contraintes spécialisées qui la constituent.

Du point de vue d'un plan de coordination sécurisée d'activités, du fait que les contraintes spécialisées constituant une contrainte locale concernent différents aspects du plan, une contrainte locale régit l'exécution de cette activité en respectant tous ces aspects du plan.

Exemple. before(Activity(*activityId*(A3a)), Activity(*activityId*(A4))) (1) est une contrainte spécialisée de l'aspect ordonnancement du plan de coordination sécurisée ; elle est à évaluer dans le cadre de l'activité de nom A3a et avant l'exécution de l'appel correspondant de cette activité.
authentic(Activity(*authenSubjProof*(ref(certA3a))), Activity(*activityId*(A3a)))) (2) est une contrainte spécialisée de l'aspect authentification du plan de coordination sécurisée ; elle est à évaluer dans le cadre de l'activité de nom A3a et avant l'exécution de l'appel correspondant de cette activité.
La conjonction de deux contraintes spécialisées (1) et (2) définit une contrainte locale de l'activité A3a. Une fois que cette contrainte est évaluée à *VRAI* , l'exécution de l'appel à une fonction d'un service peut être déclenchée.

3.2.3f Contrainte globale

Une contrainte globale correspond à des contraintes spécialisées ; ces contraintes spécialisées sont à évaluer dans le cadre de différentes activités d'un plan de coordination sécurisée et elles ne concernent qu'un aspect de ce plan. La formule qui représente une contrainte globale est construite (conformément aux règles de définition des formules bien formées, voir la section 3.2.3c) à partir de telles contraintes spécialisées et de connecteurs logiques. Dans le cas le plus simple, il s'agit de la conjonction de ces contraintes spécialisées (voir la figure 3.3). Ainsi, une contrainte globale définit des critères de qualité sur un aspect du plan. Elle exprime les relations entre les activités et/ou les services. Une contrainte globale joue le rôle de "politique" pour un aspect considéré.

Du point de vue d'une activité d'un plan de coordination sécurisée d'activités, une contrainte globale régit un aspect spécifique de l'exécution de cette activité (dans le cadre de l'exécution du plan) en fonction de certaine(s) de ses contraintes spécialisées (qui sont associées à cette activité).

Du point de vue d'un plan de coordination sécurisée d'activités, une contrainte globale régit un aspect de l'exécution de ce plan conformément à l'évaluation de toutes ses contraintes spécialisées.

Exemple. (before(Activity(*activityId*(A3a)), Activity(*activityId*(A4)))∧
before(Activity(*activityId*(A4)), Activity(*activityId*(A5))) est une contrainte globale de l'aspect ordonnancement. Elle spécifie la relation temporelle entre les trois activités A3a, A4etA5 d'un plan.

3.2.4 Activité

Une activité correspond à un appel à une fonction d'un service.

Une activité à exécuter dans le cadre d'un plan de coordination sécurisée est illustrée dans la figure 3.4. Elle est caractérisée par un identificateur (*activityId*), un ensemble des propriétés de l'activité (*properties*), un ensemble (non vide) des différents services qui participent à l'activité (*services*), un appel (*invocation*), des contraintes à satisfaire avant l'appel (*preconditions*) et des contraintes à satisfaire après l'appel (*postconditions*).

Le type Activity $\in \nabla$ caractérise des activités. Il est défini par :

$$\begin{array}{l}\mathsf{Activity} :- \mathsf{tuple}(\\ \quad \mathit{activityId} : \mathsf{String},\\ \quad \mathit{properties} : \mathsf{set}(\mathsf{Property}),\\ \quad \mathit{services} : \mathsf{set}(\mathsf{Service}),\\ \quad \mathit{invocation} : \mathsf{Invocation},\\ \quad \mathit{preconditions} : \mathsf{set}(\mathsf{Formula}),\\ \quad \mathit{postconditions} : \mathsf{set}(\mathsf{Formula})\\ )\end{array} \in \nabla \qquad (3.2.4)$$

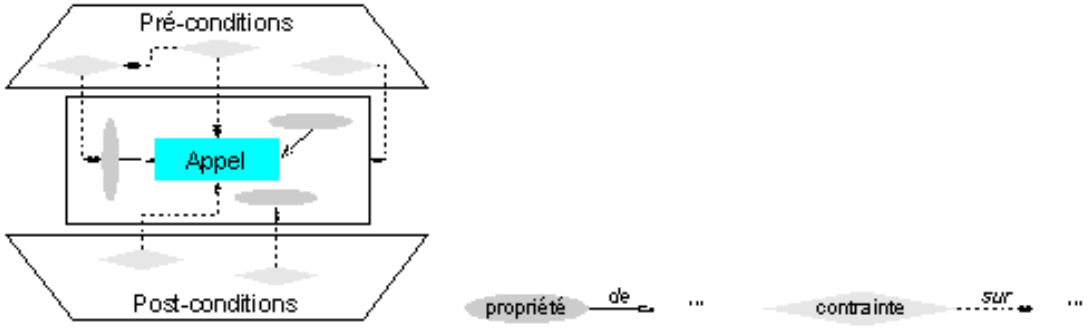


FIG. 3.4 – Activité

Un appel dans le cadre d'une activité est caractérisé par

- le nom de la fonction appelée (*iName*) ;
- des paramètres nécessaires à l'exécution de la fonction appelée conformément au mode d'emploi du service qui exécute cette fonction (*iData*) (y compris des propriétés requises par le service utilisé, ainsi que des paramètres conformément à la signature de la fonction appelée) ;
- des paramètres nécessaires à l'exécution de l'appel dans le cadre du plan de coordination (*iControl*) (y compris des propriétés de contrôle fournies par l'activité et des propriétés représentant le contexte de l'exécution de l'appel tel que le chemin d'accès à la fonction appelée, les proxies, les protocoles et les formats de messages échangés).

Les valeurs de ces paramètres sont attribuées conformément aux contraintes globales du plan. Si cela est nécessaire du point de vue du plan de coordination, l'activité attend le résultat de l'appel.

Le type Invocation caractérise des appels. Il permet de définir comment appeler une fonction d'un service. Il est défini par :

$$\begin{array}{l}\mathsf{Invocation} :\!- \mathsf{tuple}(\\ \quad \mathit{iName} : (\mathit{funcName} : \mathsf{String}),\\ \quad \mathit{iData} : \mathsf{tuple}(\\ \quad\quad \mathit{in} : \mathsf{tuple}(\mathit{funcCtrlInfo} : \mathsf{set}(\mathsf{Property}), \mathit{funcInput} : \mathsf{list}(\mathsf{T_i})),\\ \quad\quad \mathit{out} : \mathsf{tuple}(\mathit{funcCtrlInfo} : \mathsf{set}(\mathsf{Property}), \mathit{funcOutput} : \mathsf{T_j})\\ \quad),\\ \quad \mathit{iControl} : \mathsf{set}(\mathsf{Property})\\ )\end{array} \quad \in \nabla \qquad (3.2.5)$$

Exemple.

$$\begin{array}{l}\mathsf{Invocation} :\!- \mathsf{tuple}(\mathit{iName} : (\mathit{funcName} : \mathsf{selectFlight}),\\ \mathit{iData} : \mathsf{tuple}(\\ \mathit{in} : \mathsf{tuple}(\mathit{funcCtrlInfo} : (\mathit{authenObjProof} : \mathsf{ref}(\mathsf{authenSubjProofA2})), \mathit{funcInput} : \mathsf{list}(\mathsf{ref}(\mathsf{file_i}))),\\ \mathit{out} : \mathsf{tuple}(\mathit{authenObjProof} :?, \mathit{funcOutput} : \mathsf{ref}(\mathsf{file_j}))),\\ \mathit{iControl} : (\mathit{communication} : \mathsf{tuple}(\\ \mathit{proxies} : \mathsf{set}(\mathsf{ref}(\mathsf{Agence}), \mathsf{ref}(\mathsf{http} : //\mathsf{www} - \mathsf{lsr.imag.fr/HADAS} : 8080)),\\ \mathit{bindings} : \mathsf{set}(\\ \mathsf{tuple}(\mathit{protocol} : \mathsf{ref}(\mathsf{SOAPMsgSrv}), \mathit{datapackets} : \mathsf{list}(\mathit{sentmsg} : \mathsf{ref}(\mathsf{SOAPM1}), \mathit{recvmsg} : \mathsf{ref}(\mathsf{SOAPM2}))),\\ \mathsf{tuple}(\mathit{protocol} : \mathsf{ref}(\mathsf{JavaMessageService}), \mathit{datapackets} : \mathsf{list}(\mathit{sentmsg} : \mathsf{ref}(\mathsf{JMsg1}), \mathit{recvmsg} : \mathsf{ref}(\mathsf{JMsg2}))))\\ )))\end{array}$$

Ceci décrit un appel à la fonction selectFlight du service Agence dans le cadre de l'activité A2 d'un plan.

iData : une instance de cette propriété est construite à partir de valeurs typées ; ces valeurs sont conformes à la signature de la fonction selectFlight du service Agence et à son mode d'emploi. La valeur ref(authenSubjProofA2) permet de déterminer une valeur effective de la propriété d'authentification (*authenObjProof*) fournie par l'activité A2. Elle est requise par le service Agence pour l'authentification de ses utilisateurs. La valeur représentant la propriété d'authentification du service Agence est inconnue pour l'instant.

- La valeur $\mathsf{list}(\mathsf{ref}(\mathsf{file_i}))$ permet de déterminer une liste de valeurs effectives de paramètres d'entrée de la fonction appelée.
- La valeur $\mathsf{ref}(\mathsf{file_j})$ permet de déterminer la valeur effective de paramètre de sortie de la fonction appelée, une fois qu'elle a été exécutée.

iControl : une instance de propriété *communication* est requise pour contrôler l'exécution de l'appel. Cette valeur 2-uplets est construite à partir d'instances de deux propriétés *proxies* et *bindings*.

- L'instance de la propriété *proxies* (c.-à-d. la valeur $\mathsf{set}(\mathsf{ref}(\mathsf{Agence}), \mathsf{ref}(\mathsf{http} : //\mathsf{www} - \mathsf{lsr.imag.fr/HADAS} : 8080))$) présente un ensemble de chemins d'accès possible à la fonction selectFlight (resp. le nom de service fourni ou le nom de son adaptateur).
- L'instance de la propriété *bindings* présente un ensemble de connexions possibles à établir : les protocoles de communication supportés (instance de *protocols*) et les formats de messages échangés (instance de *dataPackets*).

3.2.5 Journal de coordination

Lors de l'exécution d'un plan de coordination sécurisée d'activités, un journal de coordination archive des preuves d'exécution des activités qu'il est nécessaire de produire et d'exploiter pour ce plan.

Un journal de coordination (illustré dans la figure 3.5) est caractérisé par un nom (*header*) et un ensemble des preuves d'exécution (*traces*). Les preuves d'exécution sont collectées sur les processus d'exécution des fonctions appelées. Cette collecte se fait par des services utilisés ou au nom des activités de ce plan. Nous y trouverons notamment des informations sur les fonctions appelées et sur les données échangées.

Tout journal de coordination est une instance du type :

$$\begin{array}{l}\mathsf{CoordinationLog} :\!- \mathsf{tuple}(\\ \quad \mathit{header} : \mathsf{String},\\ \quad \mathit{traces} : \mathsf{set}(\mathsf{tuple}(\\ \qquad \mathit{processTrace}, \qquad \in \nabla \\ \qquad \mathsf{list}(\mathsf{choice}(\mathit{timingTrace}, \mathit{actorTrace}, \mathit{stateTrace}, \mathit{contextTrace}, \mathit{dataTrace})))\\ \quad)\\ )\end{array} \tag{3.2.6}$$

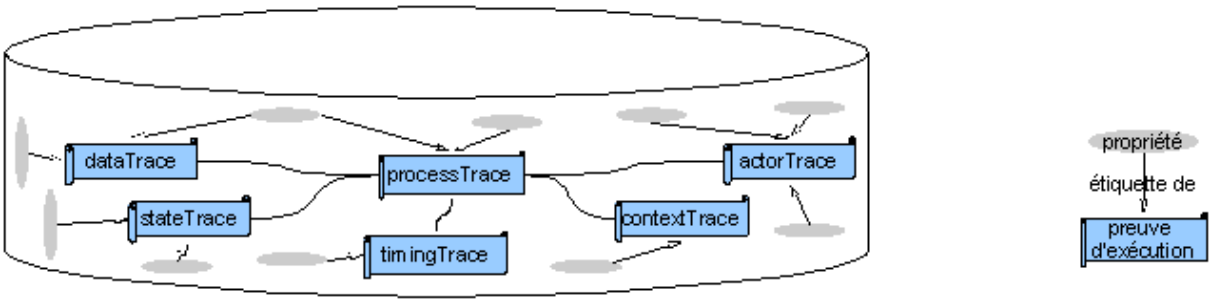


FIG. 3.5 – Journal de coordination

3.2.5a Preuve d'exécution

Une preuve d'exécution est une information qu'il est nécessaire de collecter et d'archiver pour l'exécution du plan de coordination sécurisée d'activités ou pour effectuer des analyses ultérieures. Une preuve d'exécution porte un identifiant (*traceId*) et une valeur typée (*traceVal*).

Tout type de preuves d'exécution des activités dans le cadre du plan de coordination sécurisée est décrit par le (meta) type :

$$\mathsf{Trace} :\!- \mathsf{tuple}(\mathit{traceId} : \mathsf{String}, \mathit{traceVal} : \mathsf{T}) \qquad \in \nabla \quad \text{où } \mathsf{T} \in \nabla \tag{3.2.7}$$

$\mathsf{dom}(\mathsf{Trace})$ dénote le domaine des types de preuves d'exécution, $\mathsf{dom}(\mathsf{Trace}) \subset \mathcal{D}$.

Toute preuve d'exécution remplissant les journaux de coordination est donc de type Trace.

La figure 3.5 présente les preuves d'exécution qui peuvent se produire lors de l'exécution du plan et qui peuvent être archivées dans les journaux de coordination de chaque activité. Elles

sont détaillées dans les sections suivantes. Ces preuves d'exécution sont étiquetées par les noms de certaines propriétés pour faciliter leur exploitation. Ces propriétés possèdent donc des valeurs du type Trace ; elles sont considérées comme des propriétés de non-répudiation.

Preuves de processus d'exécution Il est prévu (au moment de la spécification d'un plan de coordination sécurisée d'activités) de définir quelles sont les preuves de processus d'exécution de fonctions qui devront être produites lors de l'exécution de ce plan. Ces preuves sont collectées et enregistrées en vue d'exécuter les activités du plan.

Toute preuve de processus d'exécution est décrite par :

$$\begin{array}{l}\mathsf{Trace} :\!- \mathsf{tuple}(\\ \quad \mathit{traceId} : \mathsf{processTrace},\\ \quad \mathit{traceVal} : \mathsf{choice}(\mathsf{Operation}, \mathsf{Invocation})\\ )\end{array} \quad \in \nabla \qquad (3.2.8)$$

$\mathsf{processTrace} \rightsquigarrow \mathsf{choice}(\mathsf{Operation}, \mathsf{Invocation})$ dénote le domaine des preuves de processus d'exécution $\subset \mathsf{dom}(\mathsf{Trace})$.

Exemple.

Trace :– tuple(
traceId : processTrace,
traceVal : tuple(
operationName : login,
input : list(tuple(*username*, guest), tuple(*pwd*, foo)),
output : tuple(*result*, $VRAI$))
)

Ceci est une instance de preuve de processus d'exécution de l'opération login(guest, foo). Nous l'écrivons plus simplement :
processTrace $\rightsquigarrow$ tuple(*operationName* : login, *input* : list(guest, foo), *output* : $VRAI$)

Preuves d'acteur d'exécution Les preuves d'acteur d'exécution décrivent des services ou des activités qui prennent une part active aux processus d'exécution de fonctions dans le cadre de l'exécution des activités d'un plan. Ces preuves sont collectées et enregistrées selon des processus observés.

Toute preuve d'acteur d'exécution est décrite par :

$$\begin{array}{l}\mathsf{Trace} :\!- \mathsf{tuple}(\\ \quad \mathit{traceId} : \mathsf{actorTrace},\\ \quad \mathit{traceVal} : \mathsf{choice}(\mathsf{Service}, \mathsf{Activity})\\ )\end{array} \quad \in \nabla \qquad (3.2.9)$$

$\mathsf{actorTrace} \rightsquigarrow \mathsf{choice}(\mathsf{Service}, \mathsf{Activity})$ dénote le domaine des preuves d'acteurs d'exécution $\subset \mathsf{dom}(\mathsf{Trace})$.

Preuves de contexte d'exécution Il est nécessaire d'archiver des preuves de contexte d'exécution dans lequel les activités d'un plan sont exécutées. Il s'agit des valeurs décrivant des ser-

vices, des activités ou des propriétés faisant partie de l'environnement ambiante des processus d'exécution des fonctions dans le cadre de l'exécution des activités d'un plan. Ces valeurs sont chronologiquement collectées et enregistrées.

Toute preuve de contexte d'exécution est décrite par :

$$\begin{array}{l}\mathsf{Trace} :\!- \mathsf{tuple}(\\ \quad \mathit{traceId} : \mathsf{contextTrace},\\ \quad \mathit{traceVal} : \mathsf{choice}(\mathsf{Service}, \mathsf{Operation}, \mathsf{Property})\\ )\end{array} \in \nabla \qquad (3.2.10)$$

$\mathsf{contextTrace} \rightsquigarrow \mathsf{choice}(\mathsf{Service}, \mathsf{Operation}, \mathsf{Property})$ dénote le domaine des preuves de contexte d'exécution $\subset \mathsf{dom}(\mathsf{Trace})$.

Preuves d'ordonnancement Les preuves d'ordonnancement décrivent le temps d'exécution d'une fonction ; elles sont chronologiquement collectées et enregistrées. Elles sont de type :

$$\mathsf{Trace} :\!- \mathsf{tuple}(\mathit{traceId} : \mathsf{timingTrace}, \mathit{traceVal} : \mathsf{Time})$$

$\mathsf{timingTrace} \rightsquigarrow \mathsf{Time}$ dénote le domaine des preuves d'ordonnancement $\subset \mathsf{dom}(\mathsf{Trace})$. Les propriétés qui possèdent des valeurs appartenant à ce domaine sont aussi considérées comme des propriétés d'ordonnancement d'une activité ou d'un service.

Preuves de déclenchement Il s'agit des valeurs représentant des états d'exécution d'une fonction. Ces valeurs sont collectées et enregistrées du point de vue de la transition des états d'exécution observables. Elles sont de type :

$$\mathsf{Trace} :\!- \mathsf{tuple}(\mathit{traceId} : \mathsf{stateTrace}, \mathit{traceVal} : \mathsf{EState})$$

$\mathsf{stateTrace} \rightsquigarrow \mathsf{EState}$ dénote le domaine des preuves de déclenchement $\subset \mathsf{dom}(\mathsf{Trace})$. Les propriétés qui possèdent des valeurs appartenant à ce domaine sont aussi considérées comme des propriétés de déclenchement d'une activité ou d'un service.

Preuves de données Les preuves de données permettent d'enregistrer les valeurs de paramètres d'entrée et de sortie d'un processus exécutant une opération (fonction), une activité ou un appel au nom d'un service. Elles sont enregistrées pour permettre de visualiser les données consommées / produites pendant cette exécution. Elles sont de type :

$$\mathsf{Trace} :\!- \mathsf{tuple}(\mathit{traceId} : \mathsf{dataTrace}, \mathit{traceVal} : \mathsf{T})$$

$\mathsf{dataTrace} \rightsquigarrow \mathsf{T}$ dénote le domaine des preuves de données $\subset \mathsf{dom}(\mathsf{Trace})$. Les propriétés qui possèdent des valeurs appartenant à ce domaine sont aussi considérées comme des propriétés de données d'une activité ou d'un service.

Les propriétés de $\mathsf{dom}(\mathsf{Property})$ dont la valeur est de type Trace sont considérées comme des étiquettes qui facilitent l'exploitation de preuves d'exécution.

Exemple. Soit le domaine $\mathsf{ExecutionStep} = \{\mathsf{preparation}, \mathsf{treatment}, \mathsf{termination}\}$, qui définit les valeurs représentant différentes phases d'un processus d'exécution.
Les propriétés du type $\mathsf{Property} :\!- \mathsf{tuple}(\mathit{propertyName} : \mathsf{p}, \mathit{propertyValue} : \mathsf{Trace})$, où $\mathsf{p} \in \mathsf{ExecutionStep}$,

sont définies pour regrouper les preuves d'exécution d'opérations selon ce critère.
preparation : (*processusTrace* $\rightsquigarrow$ tuple(*operationName* : login, *input* : list(guest, foo), *output* : $VRAI$)) signifie que la preuve de processus d'exécution de l'opération login est collectée et archivée durant la phase de préparation de processus d'exécution.
treatment : (*processusTrace* $\rightsquigarrow$ tuple(*operationName* : selectFlight, *input* : list(ref(file_i)), *output* : ref(file_j))) signifie que la preuve du processus d'exécution de l'opération selectFlight est collectée et archivée lors de la phase de traitement du processus d'exécution.

3.3 Plan de coordination sécurisée

Un plan de coordination sécurisée précise les instructions à réaliser sur les activités d'une application à base de services. Lors de la spécification d'un plan de coordination sécurisée, les aspects suivants sont considérés :

- les aspects fonctionnels de coordination : ordonnancement, déclenchement et interdépendance de données.
- les aspects non-fonctionnels de sécurité : authentification, autorisation, intégrité, non-répudiation. D'autres aspects sécurité (tel que confidentialité[7]) peuvent être définis de même manière.

Ces aspects sont spécifiés à l'aide de propriétés et de prédicats prédéfinis. Par ailleurs, les aspects non-fonctionnels de contrôle de l'exécution d'un tel plan sont spécifiés en définissant au préalable les propriétés et les prédicats correspondants. De même, des aspects considérés peuvent être redéfinis en modifiant ces propriétés et prédicats.

Les instructions d'un plan de coordination sécurisée correspondent aux contraintes globales devant être conjointement satisfaites pour régir ces aspects : (i) contrainte(s) globale(s) de coordination ; (ii) contrainte(s) globale(s) de sécurité ; et éventuellement (iii) contraintes globales d'autres aspects de contrôle.

3.3.1 Ordonnancement

L'aspect ordonnancement consiste à spécifier les relations temporelles entre les activités d'un plan de coordination sécurisée.

3.3.1a Propriétés d'ordonnancement

Le domaine des propriétés d'ordonnancement $\mathit{Timing} \subset \text{dom}(\text{Property})$ se compose de trois catégories d'informations des jalons de temps d'exécution d'une activité :

- le temps de début d'exécution d'une opération dans le cadre d'une activité ou de l'activité elle-même ;
- le temps de fin d'exécution d'une opération dans le cadre d'une activité ou de l'activité elle-même ;
- la durée d'exécution d'une opération dans le cadre d'une activité ou de l'activité elle-même.

[7] Nous ne spécifions pas des propriétés et des prédicats pour l'aspect confidentialité. D'une part, selon les analyses que nous avons faits dans la section 2.5, les besoins de confidentialité ne relient pas directement aux aspects fonctionnels de coordination, mais aux informations spécifiant le plan lui-même. Donc il n'est pas nécessaire de les considérer dans la spécification du plan. D'autre part, dans notre contexte, la confidentialité des données échangées lors de l'exécution coordonnée de services est supposée d'être assurée par les services sécurité dédiés ou génériques ; ces derniers sont associés à certains services de la chaîne de services d'interaction et de réseau de l'infrastructure de communication, par exemple les protocoles chiffrés. Nous laissons donc cet aspect comme une possibilité d'extension du modèle.

Toute propriété d'ordonnancement possède des valeurs qui sont du type de base $\mathsf{Time} \in \nabla$. Le domaine des preuves d'ordonnancement ($\mathsf{timingTrace} \rightsquigarrow \mathsf{Time}$, voir la section 3.2.5a) référence une propriété d'ordonnancement dont le nom est *timingTrace* et dont les valeurs sont du type Time. De même, les propriétés qui possèdent des valeurs appartenant au domaine $\mathsf{timingTrace} \rightsquigarrow \mathsf{Time}$ sont aussi considérées comme des propriétés d'ordonnancement d'une activité ou d'un service.

3.3.1b Contraintes d'ordonnancement

Les contraintes d'ordonnancement des activités sont formulées à partir de prédicats binaires spécifiant leurs relations temporelles. Ces prédicats sont définis conformément à la quatrième règle de définition des prédicats (voir la section 3.2.3b) :

- trois termes constants `before`, `after`, `simultaneous` représentent les noms de ces prédicats ;
- les valeurs de paramètres de ces prédicats appartiennent au domaine *Timing* défini ci-dessus. Les prédicats (unaires) permettant de vérifier si une valeur appartient ou non à ce domaine sont surchargés ; cela permet de filtrer les valeurs potentiellement attribuées et de préciser la relation temporelle (entre les activités d'un plan de coordination) définie sur ces valeurs.

Étant donné deux activités $\mathsf{A_i}, \mathsf{A_j}$ de type Activity, leurs relations temporelles sont spécifiées en terme de prédicats portant sur leurs propriétés d'ordonnancement comme suit :

1. $\texttt{before}() \in \mathcal{P}$

   $\texttt{before}(\texttt{Activity}(\mathsf{A_i}), \texttt{Activity}(\mathsf{A_j}))$: $\mathsf{A_i}$ doit se produire avant $\mathsf{A_j}$[8].

   $\texttt{before}(\texttt{Activity}(\mathit{Timing}(\mathsf{t_i})), \texttt{Activity}(\mathit{Timing}(\mathsf{t_j}))) \in \mathcal{P}$: $\mathsf{A_i}$ doit se produire avant $\mathsf{A_j}$ du point de vue de leurs propriétés d'ordonnancement $\mathsf{t_i}$ et $\mathsf{t_j}$.

2. $\texttt{after}() \in \mathcal{P}$

   $\texttt{after}(\texttt{Activity}(\mathsf{A_i}), \texttt{Activity}(\mathsf{A_j}))$: $\mathsf{A_i}$ doit se produire après $\mathsf{A_j}$[9].

   $\texttt{after}(\texttt{Activity}(\mathit{Timing}(\mathsf{t_i})), \texttt{Activity}(\mathit{Timing}(\mathsf{t_j})))$: $\mathsf{A_i}$ doit se produire après $\mathsf{A_j}$ du point de vue de leurs propriétés d'ordonnancement $\mathsf{t_i}$ et $\mathsf{t_j}$.

3. $\texttt{simultaneous}() \in \mathcal{P}$

   $\texttt{simultaneous}(\texttt{Activity}(\mathsf{A_i}), \texttt{Activity}(\mathsf{A_j}))$: $\mathsf{A_i}$ et $\mathsf{A_j}$ doivent être simultanées[10].

   $\texttt{simultaneous}(\texttt{Activity}(\mathit{Timing}(\mathsf{t_i})), \texttt{Activity}(\mathit{Timing}(\mathsf{t_j})))$: $\mathsf{A_i}$ et $\mathsf{A_j}$ doivent être simultanées du point de vue de leurs propriétés d'ordonnancement $\mathsf{t_i}$ et $\mathsf{t_j}$.

Les contraintes spécialisées d'une activité $\mathsf{A_i}$ définies à l'aide de ces prédicats portent sur les propriétés d'ordonnancement de cette activité et des activités auxquelles $\mathsf{A_i}$ relie. Plus précisément, elles portent sur (voir la figure 3.6) :

- les preuves d'ordonnancement de(s) activité(s) prévue(s) pour être exécutée(s) avant $\mathsf{A_i}$; ces preuves sont extraites des spécifications de ces activités et/ou des journaux de coordination correspondants ;

  Exemple. Dans la figure 3.6 : si $\mathsf{A_j}$ est prévue pour être exécutée avant $\mathsf{A_i}$, alors la

[8] par défaut, ce prédicat est défini sur la propriété représentant le temps de fin d'exécution de $\mathsf{A_i}$ et la propriété représentant le temps de début d'exécution de $\mathsf{A_j}$.

[9] par défaut, ce prédicat est défini sur la propriété représentant le temps de début d'exécution de $\mathsf{A_i}$ et la propriété représentant le temps de fin d'exécution de $\mathsf{A_j}$.

[10] par défaut, ce prédicat est défini sur la propriété représentant la durée d'exécution de $\mathsf{A_i}$ et $\mathsf{A_j}$.

contrainte spécialisée $\texttt{after}(A_i, A_j)$ (associée à A_i) porte sur une propriété d'ordonnancement de A_i et une preuve d'ordonnancement de A_j (extraite de son journal de coordination).

- les propriétés d'ordonnancement de A_i ; elles sont extraites de la spécification de cette activité et/ou de son journal de coordination ;
- les propriétés d'ordonnancement de(s) activité(s) prévue(s) pour être exécutée(s) après A_i ou simultanément à A_i ; elles sont respectivement extraites des spécifications de ces activités ou des journaux de coordination correspondants.

Exemple. Dans la figure 3.6 : si A'_j est prévue pour être exécutée après A_i, alors la contrainte spécialisée $\texttt{before}(A_i, A'_j)$ (associée à A_i) porte sur une propriété d'ordonnancement de A_i et une preuve d'ordonnancement de A'_j (extraite de la spécification de A'_j).
Si A_j, A'_j et A_i sont prévues pour être simultanément exécutées, les contraintes spécialistes $\texttt{simultaneous}(A_i, A_j)$ et $\texttt{simultaneous}(A_i, A'_j)$ portent sur leurs propriétés d'ordonnancement (extraites de leurs spécifications).

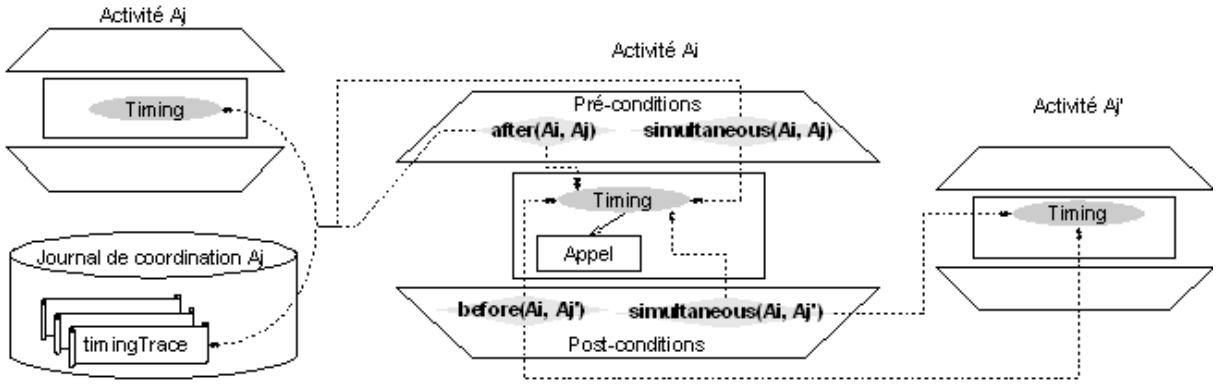


FIG. 3.6 – Portée des contraintes d'ordonnancement

3.3.2 Déclenchement

L'aspect déclenchement consiste à spécifier les relations causales entre les activités d'un plan de coordination sécurisée.

3.3.2a Propriétés de déclenchement

Le domaine $\mathsf{EState} \subset \mathcal{D}$ des états possibles d'un processus d'exécution (d'une activité, d'une fonction ou d'une contrainte, etc.) est défini par :

- des valeurs représentant des états de la phase de préparation de l'exécution : WAITING, UNDER_CONSIDERATION ;
- des valeurs représentant des états de la phase de réalisation de l'exécution : READY, ACTIVE, PAUSE, COMPENSATIVE ;
- des valeurs représentant des états de la phase de terminaison de l'exécution : COMMIT, ABORT.

Le domaine $\mathit{State} \subset \mathsf{dom}(\mathsf{Property})$ des propriétés de déclenchement se compose de toute propriété de $\mathsf{dom}(\mathsf{Property})$ et du type de base $\mathsf{EState} :- \mathsf{EState} \in \nabla$.

Le domaine des preuves de déclenchement ($\mathsf{stateTrace} \rightsquigarrow \mathsf{EState}$, voir la section 3.2.5a) référence une propriété de déclenchement dont le nom est $\mathit{stateTrace}$ et dont les valeurs sont

du type EState. De même, les propriétés qui possèdent des valeurs appartenant au domaine stateTrace $\rightsquigarrow$ EState sont aussi considérées comme des propriétés de déclenchement d'une activité ou d'un service.

3.3.2b Contraintes de déclenchement

Les contraintes de déclenchement des activités sont formulées à partir de prédicats binaires spécifiant leurs relations causales. Ces prédicats sont définis conformément à la quatrième règle de définition des prédicats (voir la section 3.2.3b). Il s'agit des prédicats binaires portant sur les propriétés de déclenchement. Les valeurs de paramètres de ces prédicats appartiennent au domaine *State* défini ci-dessus. D'abord, nous définissons quatre termes constants `waits`, `enables`, `forces`, `locks` ; ils représentent les noms de ces prédicats. Ensuite, nous définissons les prédicats vérifiant si une valeur (à attribuer à un des paramètres de prédicats de nom `waits`, `enables`, `forces`, `locks`) appartient ou non au domaine *State*. Ces prédicats sont surchargés ; cela permet de filtrer les valeurs potentiellement attribuées et de préciser la relation causale (entre les activités d'un plan de coordination) définie sur ces valeurs.

Étant donné deux activités $\mathsf{A_i}, \mathsf{A_j} \in \mathsf{dom}(\mathsf{Activity})$, leurs relations causales sont spécifiées en terme de prédicats portant sur les propriétés de déclenchement comme suit :

1. $\texttt{waits}() \in \mathcal{P}$

   $\texttt{waits}(\texttt{Activity}(\mathsf{A_i}), \texttt{Activity}(\mathsf{A_j}))$: spécifie un état de l'exécution de $\mathsf{A_i}$ qui met en attente la préparation ou la terminaison de $\mathsf{A_j}$.

   $\texttt{waits}(\texttt{Activity}(\textit{State}(\mathsf{s_i})), \texttt{Activity}(\textit{State}(\mathsf{s_j})))$: spécifie un état de l'exécution $\mathsf{s_i}$ de $\mathsf{A_i}$ qui met $\mathsf{A_j}$ en attente à l'état $\mathsf{s_j}$.

2. $\texttt{enables}() \in \mathcal{P}$

   $\texttt{enables}(\texttt{Activity}(\mathsf{A_i}), \texttt{Activity}(\mathsf{A_j}))$: spécifie un état d'exécution de $\mathsf{A_i}$ qui lance la préparation ou la terminaison de $\mathsf{A_j}$.

   $\texttt{enables}(\texttt{Activity}(\textit{State}(\mathsf{s_i})), \texttt{Activity}(\textit{State}(\mathsf{s_j})))$: spécifie un état de l'exécution $\mathsf{s_i}$ de $\mathsf{A_i}$ qui lance $\mathsf{A_j}$ avec l'état $\mathsf{s_j}$.

3. $\texttt{forces}() \in \mathcal{P}$

   $\texttt{forces}(\texttt{Activity}(\mathsf{A_i}), \texttt{Activity}(\mathsf{A_j})) \in \mathcal{P}$: spécifie un état d'exécution de $\mathsf{A_i}$ qui oblige la préparation ou la terminaison de $\mathsf{A_j}$.

   $\texttt{forces}(\texttt{Activity}(\textit{State}(\mathsf{s_i})), \texttt{Activity}(\textit{State}(\mathsf{s_j}))) \in \mathcal{P}$: spécifie un état de l'exécution $\mathsf{s_i}$ de $\mathsf{A_i}$ qui oblige $\mathsf{A_j}$ avec l'état $\mathsf{s_j}$.

4. $\texttt{locks}() \in \mathcal{P}$

   $\texttt{locks}(\texttt{Activity}(\mathsf{A_i}), \texttt{Activity}(\mathsf{A_j})) \in \mathcal{P}$: spécifie un état d'exécution de $\mathsf{A_i}$ qui bloque la préparation ou la terminaison de $\mathsf{A_j}$.

   $\texttt{locks}(\texttt{Activity}(\textit{State}(\mathsf{s_i})), \texttt{Activity}(\textit{State}(\mathsf{s_j}))) \in \mathcal{P}$: spécifie un état de l'exécution $\mathsf{s_i}$ de $\mathsf{A_i}$ qui bloque $\mathsf{A_j}$ à l'état $\mathsf{s_j}$.

Exemple. $\texttt{forces}(\texttt{Activity}(\textit{activityId}(\mathsf{A4})), \texttt{Activity}(\textit{State}(\mathsf{COMMIT})))$ est une contrainte spécialisée. Elle permet de vérifier si l'activité A4 sera terminée à l'état *COMMIT*. Les contraintes spécialisées d'une activité $\mathsf{A_i}$ définies à base de ces prédicats portent sur les propriétés de déclenchement de cette activité et des activités auxquelles $\mathsf{A_i}$ relie. Plus précisément, elles portent sur (voir la figure 3.7) :

– les preuves de déclenchement de(s) activité(s) prévue(s) pour être exécutée(s) avant A_i ; ces preuves sont extraites des journaux de coordination correspondants ;

Exemple. Dans la figure 3.7 : si A_j est prévue pour être exécutée avant A_i, alors la contrainte spécialisée `waits`(A_i, A_j) ou `locks`(A_i, A_j) (associée à A_i) portent sur une propriété de déclenchement de A_i et une preuve de déclenchement de A_j (extraite de son journal de coordination).

– les propriétés de déclenchement de A_i ; elles sont extraites de la spécification de cette activité et/ou de son journal de coordination ;

– les propriétés de déclenchement de(s) activité(s) prévue(s) pour être exécutée(s) après A_i ou parallèlement qu'A_i ; elles sont respectivement extraites de la spécification de(s) activité(s) ou de(s) journa(ux) de coordination correspondants.

Exemple. Dans la figure 3.7 : si A'_j est prévue pour être exécutée après A_i, alors la contrainte spécialisée `enables`(A_i, A'_j) ou `forces`(A_i, A'_j) (associée à A_i) porte sur une propriété de déclenchement de A_i et une preuve de déclenchement de A_j (extraite de son journal de coordination ou de son spécification).

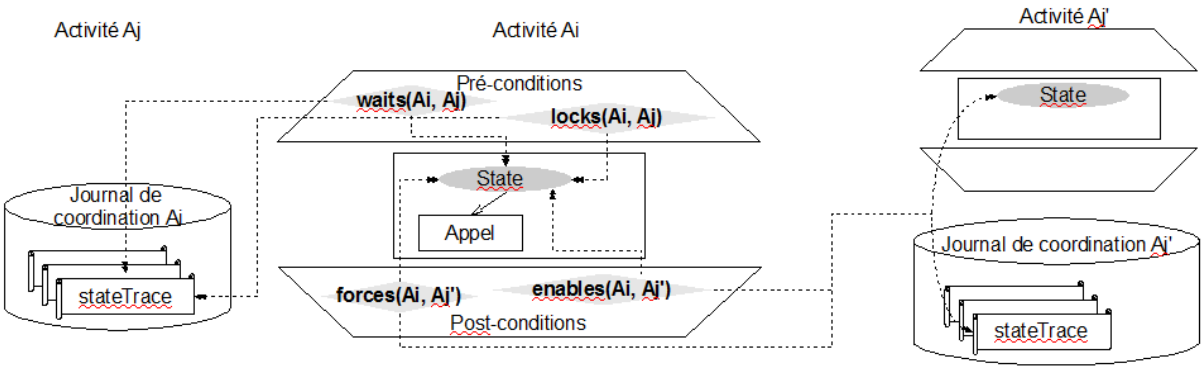


FIG. 3.7 – Portée des contraintes de déclenchement

3.3.3 Inter-dépendance de données

L'aspect inter-dépendance de données consiste à spécifier non seulement les relations subordonnées sur les informations consommées et/ou produites par les activités d'un plan de coordination sécurisée, mais aussi les restrictions sur les informations spécifiant une activité elles-même.

3.3.3a Propriétés de données

Le domaine des propriétés de données $\mathit{Data} \subset \mathsf{dom}(\mathsf{Property})$ se compose de propriétés caractérisant toute information définissant la coordination sécurisée :

– des propriétés servant à vérifier la relation subordonnée sur les informations consommées et/ou produites par des activités :

input et *output*, les propriétés caractérisant des paramètres d'entrées et de sortie d'une fonction d'un service (voir le type Operation 3.1.3) ;

iData, la propriété caractérisant des paramètres d'entrées et de sortie d'un appel (voir le type Invocation 3.2.4) ;

– des propriétés caractérisant toute information à filtrer.

Les propriétés de *Data* possèdent des valeurs de n'importe quel type de ∇.

Le domaine des preuves de données ($\mathsf{dataTrace} \rightsquigarrow \mathsf{T}$, $\mathsf{T} \in \nabla$, voir la section 3.2.5a) référence une propriété de données dont le nom est *dataTrace* et dont les valeurs sont typées. De même, les propriétés qui possèdent des valeurs appartenant au domaine de preuves de données $\mathsf{dataTrace} \rightsquigarrow \mathsf{T}$ sont aussi considérées comme des propriétés de déclenchement d'une activité ou d'un service.

3.3.3b Contraintes de données

Étant donné
deux activités $\mathsf{A_i}, \mathsf{A_j} \in \mathsf{dom}(\mathsf{Activity})$,
le type $\mathsf{T} \in \nabla$,
le type $\mathsf{Z} :- \mathsf{choice}(\mathsf{tuple}(\mathsf{T}), \mathsf{set}(\mathsf{T}), \mathsf{list}(\mathsf{T}))$ construit à base de T,
nous considérons les prédicats binaires portant sur les propriétés de *Data*. Ces prédicats sont définis conformément à la quatrième règle de définition des prédicats (voir la section 3.2.3b) comme suit :

1. $\texttt{contains}() \in \mathcal{P}$
   $\texttt{contains}(\mathsf{Z}(\mathsf{d_i}), \mathsf{T}(\mathsf{d_j})) \in \mathcal{P}$: la valeur construite $\mathsf{d_i}$ contient la valeur $\mathsf{d_j}$.
   $\texttt{contains}(\texttt{Activity}(\mathsf{Z}(\mathsf{d_i})), \texttt{Activity}(\mathsf{T}(\mathsf{d_j}))) \in \mathcal{P}$: les données de $\mathsf{A_i}$ englobent les données $\mathsf{d_j}$ de $\mathsf{A_j}$.
2. $\texttt{belongs}() \in \mathcal{P}$
   $\texttt{belongs}(\mathsf{T}(\mathsf{d_i}), \mathsf{Z}(\mathsf{d_j})) \in \mathcal{P}$: la valeur $\mathsf{d_i}$ appartient à la valeur construite $\mathsf{d_j}$.
   $\texttt{belongs}(\texttt{Activity}(\mathsf{T}(\mathsf{d_i})), \texttt{Activity}(\mathsf{Z}(\mathsf{d_j}))) \in \mathcal{P}$: les données de $\mathsf{A_i}$ sont une sous-partie de celles de $\mathsf{A_j}$.
3. $\texttt{matches}() \in \mathcal{P}$
   $\texttt{matches}(\mathsf{T}(\mathsf{d_i}), \mathsf{T}(\mathsf{d_j})) \in \mathcal{P}$: les valeurs $\mathsf{d_i}$ et $\mathsf{d_j}$ sont équivalentes.
   $\texttt{matches}(\texttt{Activity}(\mathsf{T}(\mathsf{d_i})), \texttt{Activity}(\mathsf{T}(\mathsf{d_j}))) \in \mathcal{P}$: les données de $\mathsf{A_i}$ sont équivalentes à celles de $\mathsf{A_j}$.

Exemple. La contrainte spécialisée de l'aspect inter-dépendance de données $\texttt{matches}(\texttt{Activity}(\textit{activityId}(\mathsf{A4}), \textit{output}()), \texttt{Activity}(\textit{activityId}(\mathsf{A5}), \textit{input}()))$ permet de vérifier si les données produites par l'activité A4 sont exactement les données qui seront consommées par l'activité A5.

La figure 3.8 illustre la portée des contraintes spécialisées d'une activité $\mathsf{A_i}$ qui sont définies à l'aide de ces prédicats. Dans cette figure, p représente le nom d'un des prédicats de données contains(), belongs(), matches(). Ces contraintes spécialisées portent sur les propriétés de données de cette activité et des activités $\mathsf{A_j}, \mathsf{A'_j}$ auxquelles $\mathsf{A_i}$ relie. Ces propriétés sont extraites de la spécification de ces activités et/ou de leurs journaux de coordination.

3.3.4 Authentification

L'aspect authentification spécifie la façon d'assurer que :
– l'identité de n'importe quelle activité est reconnue ;
– l'identité de n'importe quel service utilisé ou la source des informations utilisées dans le cadre du plan de coordination est reconnue ;
– les activités correspondent à des fonctions de services prévus pour être utilisés.

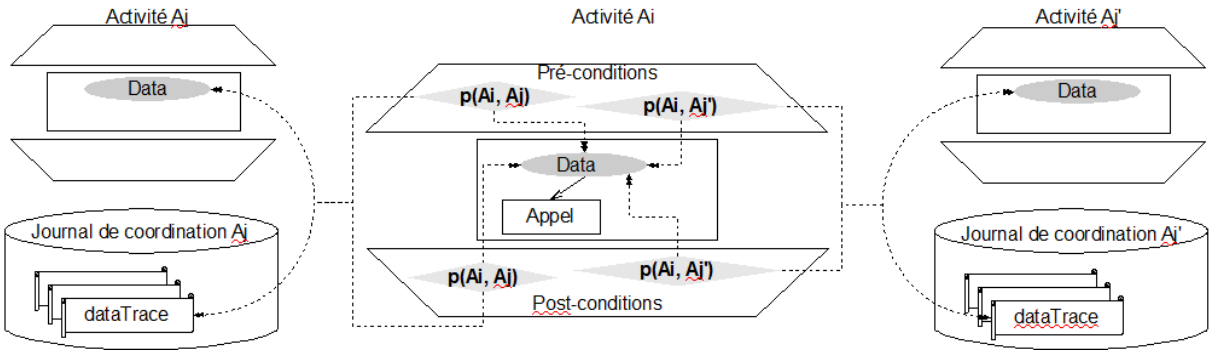


FIG. 3.8 – Portée des contraintes de données

3.3.4a Propriétés d'authentification

Le domaine $\mathit{AuthenticationPrprt} \subset \mathsf{dom}(\mathsf{Property})$ des propriétés d'authentification caractérise toute information servant à vérifier l'authentification de services, l'authentification d'activités et l'origine de données. Ce domaine est l'union de quatre domaines suivants :

- $\mathit{AuthenSubjId} \subset \mathit{AuthenticationPrprt}$: caractérise les informations qui servent à identifier les entités devant être authentifiées (identificateurs) :
- $\mathit{AuthenSubjProof} \subset \mathit{AuthenticationPrprt}$: caractérise les informations servant à démontrer l'authenticité ou l'origine des entités devant être authentifiées (preuves d'authentification) ;
- $\mathit{AuthenRule} \subset \mathit{AuthenticationPrprt}$: caractérise les informations servant à identifier et/ou à démontrer des algorithmes, des protocoles ou des schémas d'authentification ;
- $\mathit{AuthenObj} \subset \mathit{AuthenticationPrprt}$: caractérise les informations servant à identifier et/ou à démontrer (i) des entités qui font l'authentification ou (ii) des outils d'authentification utilisés et/ou supportés (p. ex. services qui fournissent des fonctions mettant en ouvre des algorithmes d'authentification).

Le domaine des preuves d'acteurs d'exécution ($\mathsf{actorTrace} \rightsquigarrow \mathsf{choice}(\mathsf{Service}, \mathsf{Activity})$, voir la section 3.2.5a) référence une propriété d'authentification. Cette propriété porte le nom $\mathit{actorTrace}$; elle possède des valeurs du type Service ou du type Activity. De même, les propriétés qui possèdent des valeurs appartenant au domaine $\mathsf{actorTrace} \rightsquigarrow \mathsf{choice}(\mathsf{Service}, \mathsf{Activity})$ sont aussi considérées comme des propriétés d'authentification.

Les propriétés d'authentification de $\mathit{AuthenticationPrprt}$ possèdent des valeurs de n'importe quel type de ∇.

3.3.4b Contraintes d'authentification

Les contraintes d'authentification sont formulées à partir de prédicats binaires portant sur les propriétés d'authentification. Les valeurs de paramètres de ces prédicats appartiennent au domaine $\mathit{AuthenticationPrprt}$ défini ci-dessus. Ces prédicats sont définis conformément à la quatrième règle de définition des prédicats (voir la section 3.2.3b). D'abord, nous définissons quatre termes constants représentant les noms de ces prédicats : `authentic`, `approval`, `coupled`, `notconsidered`. Ensuite, nous définissons les prédicats indiquant si une valeur (à attribuer à un des paramètres de prédicats `authentic`, `approval`, `coupled`, `notconsidered`) appartient ou non au domaine $\mathit{AuthenticationPrprt}$. Ces prédicats sont surchargés ; cela permet de filtrer

les valeurs potentiellement attribuées. Cela permet également de préciser la relation d'authentification entre les éléments de quatre domaines *AuthenSubjId*, *AuthenSubjProof*, *AuthenRule*, *AuthenObj*.

Étant donné $\mathsf{T} :- \mathsf{choice}(\mathsf{Service}, \mathsf{Activity}, \mathsf{Operation}, \mathsf{Invocation}) \in \nabla$, le type des entités à authentifier dans le cadre de l'exécution d'une activité, les prédicats suivants sont prédéfinis en considérant des propriétés d'authentification :

1. `authentic`() $\in \mathcal{P}$

   `authentic`(T(t), `Formula`(*AuthenRule*(a))) $\in \mathcal{P}$: spécifie qu'une instance t (de type T) à authentifier est authentifié selon l'algorithme nommé a [11] ;

   `authentic`(T(t), `Service`(*AuthenObj*(a))) $\in \mathcal{P}$: spécifie qu'une instance t (de type T) à authentifier est authentifié auprès du service de nom a ;

2. `approval`() $\in \mathcal{P}$

   `approval`(T(*AuthenticationPrprt*(t)), `Formula`(*AuthenRule*(a))) $\in \mathcal{P}$: spécifie que la propriété d'authentification t est dans la liste des propriétés d'authentification approuvées par l'algorithme nommé a [11] ;

   `approval`(T(*AuthenticationPrprt*(t)), `Service`(*AuthenObj*(a))) $\in \mathcal{P}$: spécifie que la propriété d'authentification t est dans la liste des propriétés d'authentification approuvées par le service nommé a ;

3. `coupled`(T(`list`(*AuthentificationPrprt*(t_1))), T(`list`(*AuthentificationPrprt*(t_2)))) $\in \mathcal{P}$: spécifie que les propriétés d'authentification de deux entités à authentifier se correspondent mutuellement en accord l'une avec l'autre ;

4. `notconsidered`() $\in \mathcal{P}$

   `notconsidered`(T(`AuthenticationPrprt`(t)), `Formula`(*AuthenRule*(a))) $\in \mathcal{P}$: spécifie que la propriété d'authentification t de T n'est pas considérée comme une propriété d'authentification valable pour l'algorithme nommé a [11] ;

   `notconsidered`(T(`AuthenticationPrprt`(t)), `Service`(*AuthenObj*(a))) $\in \mathcal{P}$: spécifie que la propriété d'authentification t de T n'est pas considérée comme une propriété d'authentification valable pour le service nommé a ;

Exemple. Soit authenticate1 : (*authenSubjId* : String) × (*authenSubjProof* : Certificate) → (*result* : Boolean), une fonction qui implante un algorithme d'authentification. Elle compare le nom donné (de type String) et l'identificateur extrait du certificat démontré (de type Certificate). Elle rend le résultat *VRAI* dans le cas où ces informations à comparer sont identiques, le résultat *FAUX* le cas échéant. `authentic`(`Activity`(*activityId*(A4), *authenSubjProof*(ref(certA4))), `Formula`(*authenRule*(ref(authenticate1)))) est une contrainte spécialisée de l'aspect authentification qui permet de vérifier l'authenticité de l'identificateur de l'activité A4 en utilisant cet algorithme authenticate1.

3.3.5 Autorisation

L'aspect autorisation consiste à spécifier les situations dans lesquelles :
- une activité est autorisée à s'exécuter dans le cadre du plan de coordination sécurisée ;
- un appel de fonction est autorisé dans le cadre de l'exécution d'une activité ;

[11] a est un terme fonction, voir 3.2.3a.

– une activité montre son autorisation requise pour faire exécuter une des fonctions d'un service ou pour révéler ses propriétés/contraintes à travers cet appel.

3.3.5a Propriétés d'autorisation

Le domaine $\mathsf{AuthorisationPrprt} \subset \mathsf{dom}(\mathsf{Property})$ des propriétés d'autorisation caractérise toute information servant au contrôle d'accès aux services, aux activités et aux données. Il est l'union de quatre domaines suivants :

– $\mathsf{AuthorSubjId} \subset \mathsf{dom}(\mathsf{AuthorisationPrprt})$: caractérise les autorisations fournies et requises d'un identificateur authentifié (rôles, droits, priorités ou permissions, etc.) ;
– $\mathsf{AuthorSubjProof} \subset \mathsf{dom}(\mathsf{AuthorisationPrprt})$: caractérise les informations décrivant les traces d'exécution des activités précédentes ou résultats d'évaluation des contraintes d'un plan de coordination sécurisée ;
– $\mathsf{AuthorRule} \subset \mathsf{dom}(\mathsf{AuthorisationPrprt})$: caractérise les informations servant à établir la décision de l'autorisation pour des utilisateurs identifiés ;
– $\mathsf{AuthorObj} \subset \mathsf{dom}(\mathsf{AuthorisationPrprt})$: caractérise les informations servant à identifier (i) les entités qui vérifient l'autorisation et (ii) des outils d'autorisation utilisés et/ou supportés.

Les propriétés d'autorisation de $\mathsf{dom}(\mathsf{AuthorisationPrprt})$ possèdent des valeurs de n'importe quel type de ∇.

3.3.5b Contraintes d'autorisation

Étant donné $\mathsf{T} :\!- \mathsf{choice}(\mathsf{Service}, \mathsf{Activity}, \mathsf{Operation}, \mathsf{Invocation}) \in \nabla$, le type caractérisant les entités dont l'autorisation doit être vérifiée dans le cadre de l'exécution d'une activité, les prédicats binaires suivants sont prédéfinis en considérant des propriétés d'autorisation :

1. $\mathtt{considers}(\mathsf{T}(\mathit{AuthorisationPrprt}(\mathsf{v})), \mathtt{Service}(\mathit{Operation}(\mathsf{f}))) \in \mathcal{P}$: l'appel de la fonction f, exécuté par une entité de type T (dont l'identificateur a été approuvé et qui possède la propriété d'autorisation v) dans le cadre d'une activité, est considéré.
2. $\mathtt{permits}(\mathsf{T}(\mathit{AuthorisationPrprt}(\mathsf{v})), \mathtt{Service}(\mathit{Operation}(\mathsf{f}))) \in \mathcal{P}$: l'appel de la fonction f, exécuté par une entité de type T (dont l'identificateur a été approuvée et qui possède la propriété d'autorisation v) dans le cadre d'une activité, est autorisé.
3. $\mathtt{obligates}(\mathsf{T}(\mathit{AuthorisationPrprt}(\mathsf{v})), \mathtt{Service}(\mathit{Operation}(\mathsf{f}))) \in \mathcal{P}$: l'appel de la fonction f, exécuté par une entité de type T (dont l'identificateur a été approuvée et qui possède la propriété d'autorisation v) dans le cadre d'une activité, est obligatoire.
4. $\mathtt{forbids}(\mathsf{T}(\mathit{AuthorisationPrprt}(\mathsf{v})), \mathtt{Service}(\mathit{Operation}(\mathsf{f}))) \in \mathcal{P}$: l'appel de la fonction f, exécuté par une entité de type T (dont l'identificateur a été approuvée et qui possède la propriété d'autorisation v) dans le cadre d'une activité, est interdit.

Exemple. Soit $\mathsf{selecFlight} : (\mathit{fullflightdescs} : \mathsf{list}(\mathsf{File})) \rightarrow (\mathit{chosenflightdesc} : \mathsf{File})$, une fonction fournie par le service Agence qui permet de choisir un vol à partir d'une liste de vols disponibles ; cette fonction est à appeler dans le cadre de l'activité A4.

$$\begin{aligned}&\mathtt{obligates}(\mathtt{Activity}(\mathit{activityId}(\mathsf{A4}), \mathit{authoSubjProof}(\mathsf{ref}(\mathsf{certA4}))),\\ &\qquad\mathtt{Service}(\mathit{serviceName}(\mathsf{ref}(\mathsf{Agence})), \mathtt{Operation}(\mathsf{ref}(\mathsf{selectFlight})))\\ &\qquad)\end{aligned}$$

est la contrainte spécialisée de l'aspect autorisation qui permet de vérifier (en utilisant un algorithme de l'autorisation par défaut) si l'activité A4 a les permissions pour exécuter l'appel correspondant.

Exemple. Soit checkPermissions : (*authorSubjProof* : Certificate) → (*result* : list(Permissions)), une fonction du service authoSrv1 qui implante un algorithme d'autorisation. Elle permet d'analyser le contenu d'un certificat donné et de rendre les permissions accordées qui sont décrites dans ce certificat.

```
obligates(Activity(activityId(A4), authoSubjProof(ref(certA4))),
          Service(serviceName(ref(Agence)), Operation(ref(selectFlight)),
           authoObjProof(Service(serviceName(ref(authoSrv1)), Operation(ref(checkPermissions))))
           )
          )
```

est une contrainte spécialisée de l'aspect autorisation ; cette contrainte permet de vérifier (par l'algorithme checkPermissions) si l'activité A4 a les permissions suffisantes pour exécuter l'appel correspondant.

3.3.6 Intégrité

L'aspect intégrité consiste à spécifier les situations dans lesquelles les informations, les services et les activités ne sont pas modifiés, altérés ou détruits d'une manière non autorisée, imprévue ou accidentelle. Nous considérons l'intégrité des informations décrivant le plan de coordination sécurisée, l'intégrité de l'exécution du plan lui-même (cela implique l'intégrité de l'exécution des services et des activités) et l'intégrité de messages échangés dans le cadre des activités de ce plan.

3.3.6a Propriétés d'intégrité

Le domaine *IntegrityPrprt* ⊂ dom(Property) des propriétés d'intégrité caractérise toute information servant à vérifier les modifications non autorisées des informations, des activités ou des services. Ce domaine est l'union de trois domaines suivants :

- *IntegrSubj* ⊂ *IntegrityPrprt* : ce domaine se compose de propriétés d'autorisation (propriétés de *AuthorisationPrprt*), dont en particulier celles servant à classifier les utilisateurs de services et des informations (propriétés de *AuthorRule*) ;
- *IntegrRule* ⊂ *IntegrityPrprt* : ce domaine caractérise les informations servant à établir la granularité de contrôle d'intégrité ;
- *IntegrObj* ⊂ *IntegrityPrprt* : ce domaine caractérise les informations servant à identifier (i) les entités qui vérifient l'intégrité et (ii) des outils vérifiant l'intégrité utilisés et/ou supportés.

Les propriétés d'intégrité de *IntegrityPrprt* possèdent des valeurs de n'importe quel type de ∇.

3.3.6b Contraintes d'intégrité

Étant donné

- un type T ∈ ∇ qui caractérise les entités dont l'intégrité est à vérifier dans le cadre de l'exécution d'une activité,
- dom(T), son domaine,

– un type $\mathsf{Z} \in \nabla$ qui est construit en se basant sur T,
– un prédicat $\mathtt{P} \in \mathcal{P}$,

les groupes de prédicats suivants sont prédéfinis en considérant des propriétés d'intégrité :

Le premier groupe sert à restreindre les modifications sur des informations spécifiques, par exemple la modification des données en sortie d'une activité.

1. $\mathtt{invariant}() \in \mathcal{P}$
   – $\mathtt{invariant}(\mathtt{T}(\mathsf{v}), \mathtt{IntegrityPrprt}(\mathsf{i}_2))$: la valeur v ne peut pas être modifiée après avoir été affectée ; cette vérification est basée sur la propriété d'intégrité i_2 associée.
   – $\mathtt{invariant}(\mathtt{T}(\mathsf{v}), \mathtt{Service}(\mathsf{s}))$: la valeur v ne peut pas être modifiée après avoir été affectée ; cette vérification est effectuée par le service s.
2. $\mathtt{unmodified}() \in \mathcal{P}$
   – $\mathtt{unmodified}(\mathtt{list}(\mathtt{Activity}(\mathtt{T}(\mathsf{v}_1)), \mathtt{Activity}(\mathtt{T}(\mathsf{v}_2))), \mathtt{Service}(\mathsf{s}))$: le contenu de message échangé entre $\mathsf{A_i}$ et $\mathsf{A_j}$ ne peut pas être modifié ; cette vérification est effectuée par le service s.
   – $\mathtt{unmodified}(\mathtt{Activity}(\mathtt{T}(\mathsf{v}_1), \mathtt{IntegrityPrprt}(\mathsf{i})), \mathtt{Activity}(\mathtt{T}(\mathsf{v}_2), \mathtt{IntegrityPrprt}(\mathsf{j})))$: le contenu de type T d'un message échangé (ou le message échangé lui-même) entre deux activités ne peut pas être modifié ; cette vérification est basée sur des propriétés d'intégrité fournies par les activités concernées.

Le deuxième groupe sert à restreindre les valeurs potentiellement affectées.

1. $\mathtt{unique}(\mathtt{T}(\mathsf{t}), \mathtt{Z}(\mathsf{z})) \in \mathcal{P}$: spécifie que la valeur t de type T constituant la valeur construite z de type Z est unique parmi des valeurs de type T constituant des valeurs de domaine $\mathsf{dom}(\mathsf{Z})$.
2. $\mathtt{range}(\mathtt{T}(\mathsf{t}), \mathtt{T}'(\mathsf{t}), \mathtt{Z}(\mathsf{z})) \in \mathcal{P}$: spécifie que la valeur t de type T constituant la valeur construite z de type Z, ainsi que toute valeur de type T constituant des valeurs de domaine $\mathsf{dom}(\mathsf{Z})$, appartient au domaine $\mathsf{dom}(\mathsf{T}') \subseteq \mathsf{dom}(\mathsf{T})$.
3. $\mathtt{free}(\mathtt{T}(\mathsf{t}), \mathtt{Z}(\mathsf{z})) \in \mathcal{P}$: spécifie qu'il n'y a aucune restriction sur la valeur t de type T constituant la valeur construite z de type Z, ainsi que sur toute valeur de type T constituant des valeurs de domaine $\mathsf{dom}(\mathsf{Z})$.

Le troisième groupe sert à définir la validité de valeurs constituant des informations spécifiques.

1. $\mathtt{wellformed}(\mathtt{T}(\mathsf{t}), \mathtt{P}(\mathsf{T})) \in \mathcal{P}$: spécifie que toute valeur t de $\mathsf{dom}(\mathsf{T})$ est correctement structurée conformément au prédicat $\mathtt{P}(\mathsf{T})$.
2. certains connecteurs et quantificateurs d'une formule logique ($\nexists, \exists, \forall$) sont explicitement récrits en termes prédicats d'intégrité afin d'éviter des ambiguës éventuelles lors de la définition des contraintes de différents aspects :
   – $\mathtt{nexists}(\mathtt{T}(\mathsf{t}), \mathtt{P}(\mathtt{T}(\mathsf{t}))) \in \mathcal{P}$: spécifie que aucune valeur t de $\mathsf{dom}(\mathsf{T})$ ne satisfait le prédicat $\mathtt{P}(\mathtt{T}(\mathsf{t}))$;
   – $\mathtt{exists}(\mathtt{T}(\mathsf{t}), \mathtt{P}(\mathtt{T}(\mathsf{t}))) \in \mathcal{P}$: spécifie qu'il n'existe qu'une seule valeur t de $\mathsf{dom}(\mathsf{T})$ qui satisfait le prédicat $\mathtt{P}(\mathtt{T}(\mathsf{t}))$;
   – $\mathtt{forall}(\mathtt{T}(\mathsf{t}), \mathtt{P}(\mathtt{T}(\mathsf{t}))) \in \mathcal{P}$: spécifie que toute valeur t de $\mathsf{dom}(\mathsf{T})$ satisfait le prédicat $\mathtt{P}(\mathtt{T}(\mathsf{t}))$.

Exemple. $\mathtt{unmodified}(\mathtt{Activity}(\mathit{activityId}(\mathsf{A4}), \mathit{output}()), \mathtt{Activity}(\mathit{activityId}(\mathsf{A5}), \mathit{input}()))$ est une contrainte spécialisée de l'aspect intégrité qui permet de vérifier si les données échangées entre les activités $\mathsf{A4}$ et $\mathsf{A5}$ sont modifiées.

3.3.7 Non-répudiation

L'aspect non-répudiation consiste à spécifier la façon de produire et d'analyser les preuves démontrant que :

- les activités n'ont pas enfreint le plan ;
- les activités ayant subi une attaque, les informations ou services étant compromis à une menace effective sont ultérieurement détectés ;
- un service a effectué une activité de manière certaine.

3.3.7a Propriétés de non-répudiation

Le domaine $\mathit{NonRepudiationPrprt} \subset \mathsf{dom}(\mathsf{Property})$ des propriétés de non-répudiation caractérise toute information servant à vérifier les modifications non autorisées des informations, des activités ou des services. Ce domaine est l'union de trois domaines des propriétés suivants :

- $\mathit{NonRepSubj} \subset \mathit{NonRepudiationPrprt}$: ce domaine se compose :

  des preuves d'exécution des activités ;

  des propriétés caractérisant des preuves d'exécution des activités ;

  des propriétés d'authentification de $\mathit{AuthenObj}$ ou de $\mathit{AuthenSubjId}$;

  des propriétés d'autorisation de $\mathit{AuthorObj}$ ou de $\mathit{AuthorSubjId}$;

  des propriétés d'intégrité de $\mathit{IntegrObj}$.

- $\mathit{NonRepRule} \subset \mathit{NonRepudiationPrprt}$: caractérise les informations servant à établir la relation entre des algorithmes d'audit, d'imputation et de traçage.
- $\mathit{NonRepObj} \subset \mathit{NonRepudiationPrprt}$: caractérise les informations servant à identifier (i) les entités qui vérifient la non-répudiation et (ii) des outils effectuant l'audit, l'imputation, le traçage et la prévention contre des services causés des comportements non souhaités.

Les propriétés de non-répudiation de $\mathit{NonRepudiationPrprt}$ possèdent des valeurs de n'importe quel type de ∇.

3.3.7b Contraintes de non-répudiation

Étant donné $\mathsf{T} :\!- \mathsf{choice}(\mathsf{Activity}, \mathsf{Operation}, \mathsf{Invocation}) \in \nabla$, les prédicats suivants sont définis en se basant principalement sur les données extraites des journaux de coordination :

1. $\mathtt{ordered}(\mathtt{Activity}(\mathsf{A_i})) \in \mathcal{P}$: spécifie que l'on peut déterminer l'ordre d'exécution réel de l'activité $\mathsf{A_i}$; cet ordre est exactement l'ordre décrit par les contraintes de l'aspect ordonnancement. Cela est fait en utilisant les preuves constituant l'histoire de l'évaluation des contraintes associées à $\mathsf{A_i}$, celles-ci étant récupérées du journal de coordination correspondant.
2. $\mathtt{approving}(\mathtt{Activity}(\mathsf{A_i})) \in \mathcal{P}$: spécifie que l'on peut déterminer qui est responsable de la validation des données échangées dans le cadre de l'activité $\mathsf{A_i}$; ce responsable est exactement ce qui est prévu en terme des propriétés d'authentification. Cela est fait en utilisant les preuves constituant l'histoire d'exécution de $\mathsf{A_i}$, celles-ci étant récupérées du journal de coordination correspondant.
3. $\mathtt{sent}(\mathtt{Activity}(\mathsf{A_i})) \in \mathcal{P}$: spécifie que l'on peut déterminer qui a envoyé les données (c.-à-d. les paramètres d'un appel ou d'un résultat) dans le cadre de l'activité $\mathsf{A_i}$; celui-ci est exactement le service émetteur qui est décrit dans le plan de coordination. Cela est fait en

utilisant les preuves constituant l'histoire d'exécution de $\mathsf{A_i}$, celles-ci étant récupérées du journal de coordination correspondant. Les propriétés d'authentification et d'autorisation de $\mathsf{A_i}$ contribuent aussi à cela.

4. $\texttt{original}(\texttt{Activity}(\mathsf{A_i})) \in \mathcal{P}$: spécifie que l'on peut prouver à la fois qui a envoyé les données et qui les a approuvé dans le cadre de l'activité $\mathsf{A_i}$.
5. $\texttt{received}(\texttt{Activity}(\mathsf{A_i})) \in \mathcal{P}$: spécifie que l'on peut prouver le fait que le service destinataire a bien reçu les données (c.-à-d. les paramètres d'un appel ou d'un résultat) dans le cadre de l'activité $\mathsf{A_i}$. Cela est fait en utilisant les preuves constituant l'histoire d'exécution de $\mathsf{A_i}$, celles-ci étant récupérées du journal de coordination correspondant.
6. $\texttt{known}(\texttt{Activity}(\mathsf{A_i})) \in \mathcal{P}$: spécifie que l'on peut prouver le fait que le service destinataire a eu connaissance du contenu de ce qu'il a reçu (c.-à-d. les paramètres d'un appel ou d'un résultat) dans le cadre de l'activité $\mathsf{A_i}$.
7. $\texttt{delivered}(\texttt{Activity}(\mathsf{A_i})) \in \mathcal{P}$: spécifie que l'on peut prouver le fait que le service destinataire a reçu les données et qu'il a eu connaissance de leur contenu.
8. $\texttt{submitted}(\texttt{Activity}(\mathsf{A_i})) \in \mathcal{P}$: spécifie que l'on peut prouver le fait qu'une autorité de la livraison a accepté le transfert des données (c.-à-d. les paramètres d'un appel ou d'un résultat).
9. $\texttt{transported}(\texttt{Activity}(\mathsf{A_i})) \in \mathcal{P}$: spécifie que l'on peut prouver le fait qu'une autorité de la livraison a transféré les données (c.-à-d. les paramètres d'un appel ou d'un résultat) d'un service expéditeur spécifique à un service destinataire prévu.

Exemple.

$$\texttt{ordered}(\texttt{Activity}(\textit{activityId}(\mathsf{A4}),\ \textit{preconditions}(\texttt{before}(\texttt{Activity}(\textit{activityId}(\mathsf{A3a})), \texttt{Activity}(\textit{activityId}(\mathsf{A4}))))))$$

est une contrainte spécialisée de l'aspect non-répudiation qui permet de vérifier a posteriori (en se basant sur le journal de coordination des activités A3a et A4a) si l'activité A4 a été exécutée dans un ordre bien établi.

3.3.8 D'autres aspects contrôle

Nous introduisons les aspects suivants autour de l'exécution du plan de coordination sécurisée de services. Certaines propriétés et contraintes de chaque aspect seront détaillées dans le chapitre4 décrivant le modèle d'évaluation des propriétés et contraintes.

3.3.8a Transformation

Cet aspect consiste à transformer la spécification des contraintes globales d'un plan de coordination sécurisée en un plan exécutable par des éléments constitutifs d'une architecture d'exécution.

Le domaine $\mathsf{renderingPrprt} \subset \mathsf{dom}(\mathsf{Property})$ contient des propriétés qui caractérisent toute information utile pour cette transformation.

Les prédicats prédéfinis pour cet aspect contribuent à définir des règles de transformation des contraintes, par exemple des règles pour vérifier a priori leur cohérence, leur redondance, etc.

3.3.8b Sûreté de fonctionnement

Les propriétés et prédicats de cet aspect servent à la sûreté de fonctionnement de l'exécution du plan.

Le domaine $\mathsf{safetyPrprt} \subset \mathsf{dom}(\mathsf{Property})$ contient des propriétés qui caractérisent toute information utile pour cette sûreté de fonctionnement.

3.3.8c Adaptation

Les propriétés et prédicats de cet aspect servent à l'adaptation de l'exécution du plan.

Le domaine $\mathsf{adaptingPrprt} \subset \mathsf{dom}(\mathsf{Property})$ contient des propriétés qui caractérisent toute information utile pour cette adaptation.

3.4 Utilisation du modèle

En utilisant le modèle MEO, une application à base de services est vue comme un plan de coordination sécurisée d'activités. La spécification de cette application consiste à :

- identifier les activités à exécuter : nom, appel, services utilisés, propriétés étant utiles à l'exécution de l'appel ;
- identifier les aspects considérés de l'exécution du plan (voir la section 3.3) : contraintes globales de ces aspects qui doivent être conjointement satisfaites, les propriétés servant au contrôle, à la gestion et à l'évaluation de ces contraintes ;
- associer ces contraintes et ces propriétés aux activités pour compléter la spécification des activités.

Cette section décrit d'abord les règles d'association entre des contraintes et des activités. Ces règles permettent de définir (i) où une contrainte doit être évaluée, et (ii) quand elle doit être satisfaite. Ensuite, cette section montre à travers un exemple de définition de plan de coordination sécurisée comment tous les concepts du modèle sont utilisés.

3.4.1 Guide d'association entre des contraintes et des activités

Les contraintes d'un plan de coordination sécurisée d'activités sont définies par des formules bien formées. De manière générale, elles sont associées aux activités selon les règles suivantes :

1. Une contrainte spécialisée est associée comme une condition d'un de types suivants :
   - **Pré** : Si une contrainte spécialisée doit être satisfaite avant l'exécution d'un appel dans le cadre d'une activité spécifique, alors elle est associée à cette activité en tant que pré-condition de cet appel.
   - **Post** : Si une contrainte spécialisée doit être satisfaite après l'exécution d'un appel dans le cadre d'une activité spécifique, alors elle est associée à cette activité en tant que post-condition de l'appel correspondant.
   - **Pré + Post** : Si une contrainte spécialisée doit être satisfaite pendant l'exécution d'un appel dans le cadre d'une activité spécifique, alors elle est associée à cette activité en tant qu'à la fois pré-condition et post-condition de l'appel correspondant.
2. Une contrainte locale (correspond aux contraintes spécialisées à évaluer conjointement) est associée à l'activité à laquelle toutes les contraintes spécialisées constituantes sont associées en tant que propriété de sûreté de fonctionnement de cette activité.

3. Une contrainte globale (correspond aux contraintes spécialisées concernant le même aspect du plan) est associée à toutes activités auxquelles les contraintes constituantes sont associées en tant que propriété de sûreté de fonctionnement de ces activités.

Étant données
- A_i, A_j, A_k, trois activités construisant une application à base de services ;
- t_i, une information (de type T) appartenant à la spécification de l'activité A_i ou au journal de coordination correspondant de cette activité ;
- t_j, une information (de type T) appartenant à la spécification de l'activité A_j ou au journal de coordination correspondant de cette activité ;
- t_k, une information (de type T) appartenant à la spécification de l'activité A_k ou au journal de coordination correspondant de cette activité ;
- $p(\texttt{Activity}(T(t_i)), \texttt{Activity}(T(t_j)), .., \texttt{Activity}(T(t_k)))$, un prédicat n-aire de $\mathcal{P}$;

les contraintes de différents aspects sont associées à ces activités selon les règles suivantes (voir la figure 3.9) :

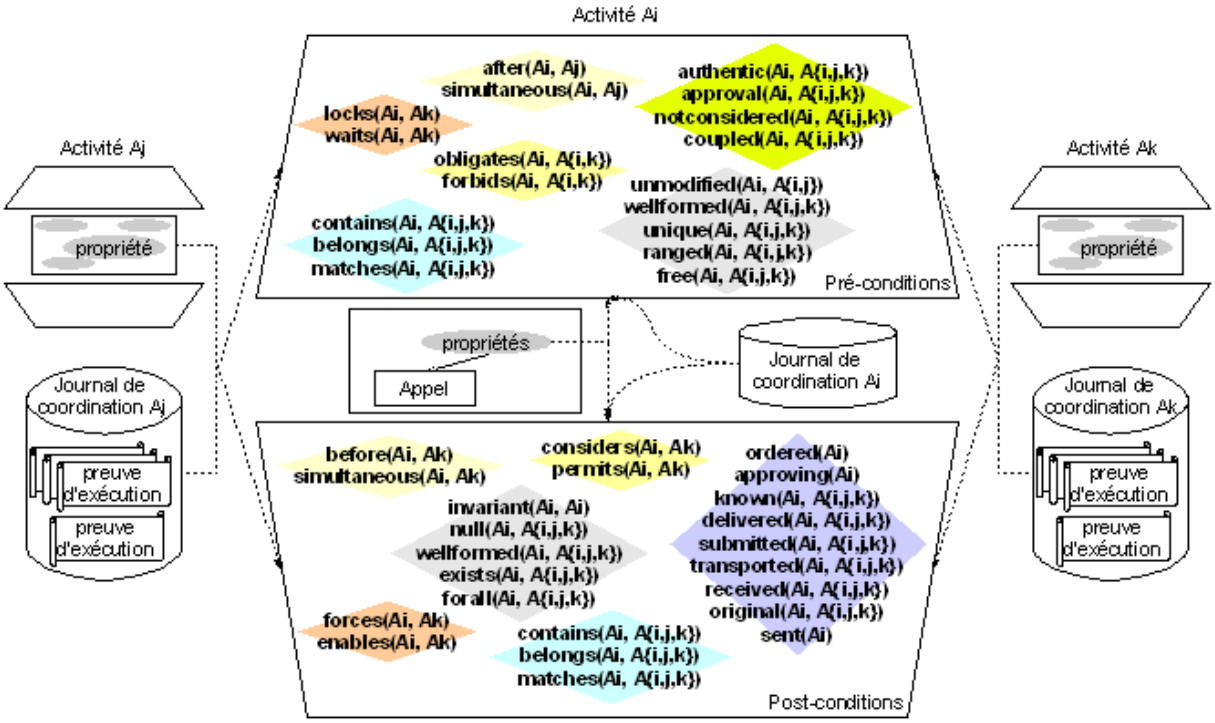


FIG. 3.9 – Règles d'association des contraintes de différents aspects aux activités

1. Une contrainte spécialisée à base d'un prédicat n-aire $p(\texttt{Activity}(T(t_i)), \texttt{Activity}(T(t_j)), .., \texttt{Activity}(T(t_k)))$ est associée à l'activité A_i.
2. Pour l'aspect d'ordonnancement :
   - Une contrainte spécialisée à base d'un des prédicats `after`(), `simultaneous`() **peut** être définie comme la pré-condition de l'appel dans le cadre de l'activité à laquelle elle est associée.
   - Une contrainte spécialisée à base d'un des prédicats `before`(), `simultaneous`() **peut** être définie comme la post-condition de l'appel dans le cadre de l'activité à laquelle elle est associée.
3. Pour l'aspect déclenchement :

- Une contrainte spécialisée à base d'un des prédicats `locks`(), `waits`() **peut** être définie comme la pré-condition de l'appel dans le cadre de l'activité à laquelle elle est associée.
- Une contrainte spécialisée à base d'un des prédicats `enables`(), `forces`() **peut** être définie comme la post-condition de l'appel dans le cadre de l'activité à laquelle elle est associée.

4. Pour l'aspect inter-dépendance de données, une contrainte spécialisée à base d'un des prédicats `contains`(), `belongs`(), `matches`() **peut** être définie comme la pré-condition ou la post-condition de l'appel dans le cadre de l'activité à laquelle elle est associée.
5. Pour l'aspect authentification, une contrainte spécialisée à base d'un des prédicats `coupled`(), `authentic`(), `approval`(), `notconsidered`() **peut** être définie comme la pré-condition de l'appel dans le cadre de l'activité à laquelle elle est associée.
6. Pour l'aspect autorisation :
   - Une contrainte spécialisée à base d'un des prédicats `locks`(), `waits`() **peut** être définie comme la pré-condition de l'appel dans le cadre de l'activité à laquelle elle est associée.
   - Une contrainte à base d'un des prédicats `enables`(), `forces`() **peut** être définie comme la post-condition de l'appel dans le cadre de l'activité à laquelle elle est associée.
7. Pour l'aspect integrité :
   - Une contrainte spécialisée à base d'un des prédicats `unmodified`(), `wellformed`(), `unique`(), `ranged`(), `free`() **peut** être définie comme la pré-condition de l'appel dans le cadre de l'activité à laquelle elle est associée.
   - Une contrainte spécialisée à base d'un des prédicats `invariant`(), `wellformed`(), `nexists`(), `exists`(), `forall`() **peut** être définie comme la post-condition de l'appel dans le cadre de l'activité à laquelle elle est associée.
8. Pour l'aspect non-répudiation, une contrainte spécialisée à base d'un des prédicats `ordered`(), `submitted`(), `approving`(), `original`(), `received`(), `sent`(), `known`(), `delivered`(), `transported`() **peut** être définie comme la post-condition de l'appel dans le cadre de l'activité à laquelle elle est associée. Elle **peut** être également redéfinie comme la pré-condition de l'appel dans le cadre de l'activité qui doit s'exécuter après l'activité à laquelle elle est associée conformément à la règle 1.
9. Pour d'autres aspects de contrôle, une contrainte spécialisée à base d'un des leurs prédicats prédéfinis **peut** être définie comme la pré ou la post-condition de l'appel dans le cadre de l'activité à laquelle elle est associée.

3.4.2 Exemple d'utilisation

L'exemple de l'utilisation du modèle consiste en la spécification d'un plan de coordination sécurisée pour construire une application de réservation de vols à partir de deux services existants. En appelant des fonctions de services existants, cette application effectue les fonctions suivantes :

- chercher une liste des places disponibles selon les besoins de clients ;
- réserver une place en choisissant à partir de la liste des places disponibles ;
- acheter un billet.

Dans cette section, nous montrons :

- la description des services utilisés (voir la section 3.4.2a) ;
- la logique de l'application que nous voulons construire (voir la section 3.4.2b) ;

– la description des activités du plan (voir la section 3.4.2c) ;
– les politiques de sécurité appliquées (voir la section 3.4.2e) ;
– les propriétés fournies par des activités (voir la section 3.4.2f).

3.4.2a Descriptions de services utilisés

Agence Le service *Agence* fournit des fonctions (voir FIG. 3.10) permettant l'interaction et la gestion des clients qui s'intéressent à la réservation des places ; la recherche des places disponibles ainsi la pré-réservation des places choisies sur des bases de données de vols de différents opérateurs.

signatures : set(
login : (*username* : String) × (*pwd* : String) → (*result* : Boolean),
selectFlight : (*fullflightdescs* : list(File)) → (*chosenflightdesc* : File),
displayResult : (*paymentRef* : String) × (*bookingRef* : String) → (*confirmation* : list(String)),
seekFlights : (*destination* : String) × (*date* : String) → (*availableflightlist* : list(File)),
showReservationDialog : (*chosenflightdesc* : File) → (*chosenflightdesc* : File) × (*clientRef* : String),
prebook : (*clientRef* : String) × (*chosenflightdesc* : File) → (*bookingRef* : String),
cancelReservation : (*clientRef* : String) × (*bookingRef* : String) → (*confirmation* : String)
)

FIG. 3.10 – Fonctions à coordonner, fournies par le service Agence

Le mode d'emploi de ce service spécifie les aspects fonctionnels et non-fonctionnels de son exécution : sa preuve de contrôle d'accès, la visibilité de ses fonctions, leur ordre d'appel à respecter ainsi le service d'interaction utilisé lors de la communication, etc. FIG. 3.11 représente une partie de son mode d'emploi :

properties : set(
authorSubjProof : Certificate :– ref(certificate1),
communication : tuple(
proxies : set(Service :– ref(Agence), Service :– ref(http : //www − lsr.imag.fr/HADAS : 8080)),
binding : set(
tuple(*protocol* : Service :– ref(JavaMessageService),
datapackets : list(*sentmsg* : MsgFormat :– ref(JMsgFormat1), *recvmsg* : MsgFormat :– ref(JMsgFrmt2))),
tuple(*protocol* : Service :– ref(SOAPMessageService),
datapackets : list(
sentmsg : MsgFormat :– ref(SOAPMsgFrmt1), *recvmsg* : MsgFormat :– ref(SOAPMsgFrmt2)))))
),
constraints : set(
`Formula` :– public(set(ref(login), ref(displayResult), ref(selectFlight))),
`Formula` :– after(ref(selectFlight), set(ref(login), ref(displayResult))),
`Formula` :– required(Certificate(), Service(*authenObjProof*()))
)

FIG. 3.11 – Extrait du mode d'emploi du service Agence : contraintes et propriétés à reconnaître au préalable

Bank Le service *Bank* fournit des fonctions (voir la figure 3.12) permettant de vérifier le solde d'un compte bancaire d'un client, de débiter le compte bancaire d'un montant ou de créditer un montant à un compte bancaire d'un client. Ces fonctions intègrent l'authentification du titulaire de ce compte.

3.4.2b Logique applicative

La figure 3.13 illustre une logique applicative simple, décrite comme suit :

signatures : set(
checkAccountBalance : (*clientRef* : String) × (*amount* : Float) → (*authorisation* : Boolean),
debit : (*clientRef* : String) × (*amount* : Float) → (*paymentRef* : String),
credit : (*clientRef* : String) × (*amount* : Float) → (*paymentRef* : String)
)

FIG. 3.12 – Fonctions à coordonner, fournies par le service Bank

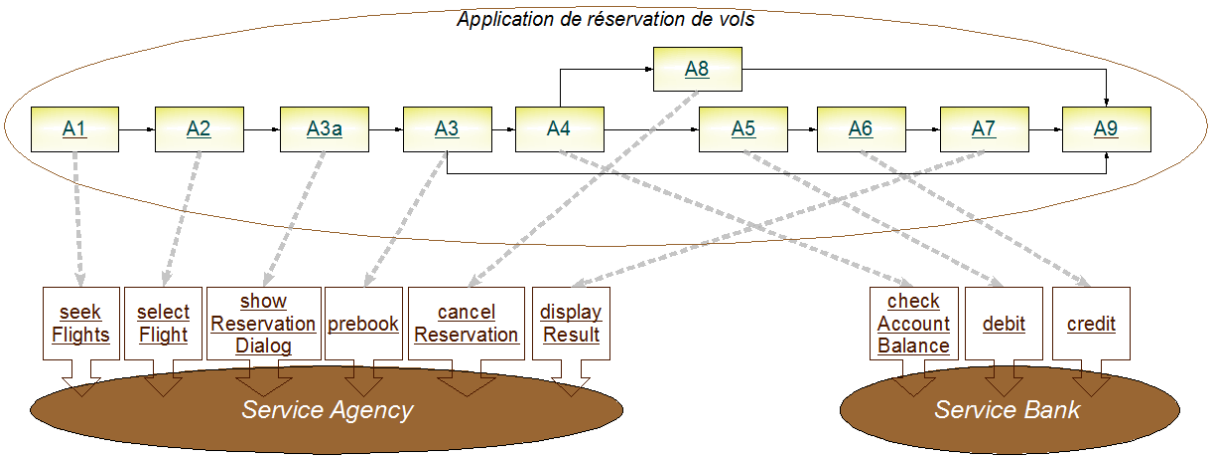


FIG. 3.13 – Application de réservation de vols : logique applicative

- L'application de réservation de vols interagit avec service *Agence* pour rechercher des vols disponibles en appelant la fonction nommée seekFlights. Ce service renvoie une liste d'informations sur des vols possibles. Cela correspond à l'exécution de l'activité A1.
- Une fois qu'un vol dans cette liste est choisi par le client, l'application de réservation de vols appelle alors la fonction nommée selectFlight du service *Agence* pour obtenir des informations détaillées sur le vol choisi. Cela correspond à l'exécution de l'activité A2.
- L'application de réservation de vols appelle ensuite la fonction nommée showReservationDialog du service *Agence* en utilisant l'information sur le vol choisi. Ce service renvoie un panier où l'information sur le vol a été pré remplie et les coordonnées du client sont à compléter. Cela correspond à l'exécution de l'activité A3a.
- Une fois que le client remplit ses coordonnées et choisit de réserver le vol, l'information sur le client et sur le vol choisi est alors dirigée au service *Agence*. Ce service effectue des opérations de pré-réservation sur une base de données de vol par la fonction prebook. La confirmation de pré-réservation impliquant l'information d'achat est retournée. Cela correspond à l'exécution de l'activité A3.
- Une fois que le client a choisi le paiement, l'application appelle la fonction checkAccountBalance du service *Bank* pour avoir l'autorisation de paiement de la banque. Cela correspond à l'exécution de l'activité A4. A ce moment il y a deux possibilités.
- Si le paiement est accepté :
  - L'application appelle la fonction debit du service *Bank* pour débiter le compte bancaire du client d'un montant correspondant à ce paiement (cela correspond à l'exécution de l'activité A5), et
  - Elle appelle la fonction credit de même service pour créditer ce montant au compte bancaire du service d'agence touristique (cela correspond à l'exécution de l'activité A6), et finalement
  - La réservation est complète en fonction d'une confirmation délivrée après l'appel de la

fonction displayResult du service *Agence*. Cela correspond à l'exécution de l'activité A7.

- Si le paiement n'est pas accepté, l'application annule la réservation en appelant la fonction cancelReservation du service *Agence*. Cela correspond à l'exécution de l'activité A8.
- La session de réservation est fermée en appelant la fonction close du service *Agence*. Cela correspond à l'exécution de l'activité A9.

3.4.2c Activités à coordonner

Des activités suivantes sont déclarées pour construire cette application :

A1, pour afficher la liste des informations sur des vols disponibles (voir Fig. 3.14) ;

A2, pour récupérer des informations sur le vol choisi par le client (voir Fig. 3.15) ;

```
Activity :– tuple(activityId : A1, properties :?,
 services : set(ref(targetApp), ref(Agence), ?),
 invocation : tuple(iName : seekFlights,
  iData : tuple(
   in : tuple(funcCtrlInfo : (sender :?),
    funcInput : list(destination :?, date :?)),
   out : tuple(funcCtrlInfo : (sender :?),
    funcInput : list(availableflightslist :?))),
   iControl : (communication : tuple(
    proxies : ref(Agence),
    bindings : tuple(protocols : ref(invoke),
    dataPackets : φ)))),
  preconditions :?, postconditions :?
 )
```

Fig. 3.14 – Activité A1

```
Activity :– tuple(activityId : A2, properties :?,
 services : set(ref(targetApp), ref(Agence), ?),
 invocation : tuple(iName : selectFlight,
  iData : tuple(
   in : tuple(funcCtrlInfo : (sender :?),
    funcInput : list(fullflightdescs :?)),
   out : tuple(funcCtrlInfo : (sender :?),
    funcOutput : list(chosenflightdesc :?))),
   iControl : (communication : tuple(
    proxies : ref(Agence),
    bindings : tuple(protocols : ref(invoke),
    dataPackets : φ)))),
  preconditions :?, postconditions :?
 )
```

Fig. 3.15 – Activité A2

A3a, pour afficher le panier (voir Fig. 3.16) ;

A3, pour effectuer la pré-réservation du vol choisi (voir Fig. 3.17) ;

```
Activity :– tuple(activityId : A3a, properties :?,
 services : set(ref(targetApp), ref(Agence), ?),
 invocation : tuple(iName : showReservationDialog,
  iData : tuple(
   in : tuple(funcCtrlInfo : (sender :?),
    funcInput : list(chosenflightdesc :?)),
   out : tuple(funcCtrlInfo : (sender :?),
    funcOutput : list(chosenflightdesc :?, clientRef :?))),
   iControl : (communication : tuple(
    proxies : ref(Agence),
    bindings : tuple(protocols : ref(invoke),
    dataPackets : φ)))),
  preconditions :?, postconditions :?
 )
```

Fig. 3.16 – Activité A3a

```
Activity :– tuple(activityId : A3, properties :?,
 services : set(ref(targetApp), ref(Agence), ?),
 invocation : tuple(iName : prebook,
  iData : tuple(
   in : tuple(funcCtrlInfo : (sender :?),
    funcInput : list(clientRef :?, chosenflightdesc :?)),
   out : tuple(funcCtrlInfo : (sender :?),
    funcOutput : list(bookingRef :?))),
   iControl : (communication : tuple(
    proxies : ref(Agence),
    bindings : tuple(protocols : ref(invoke),
    dataPackets : φ)))),
  preconditions :?, postconditions :?
 )
```

Fig. 3.17 – Activité A3

A4, pour vérifier le solde du compte bancaire du client (voir Fig. 3.18) ;

A5, pour débiter le paiement à partir du compte bancaire du client (voir Fig. 3.19) ;
A6, pour créditer le paiement au compte bancaire de l'agence touristique (voir Fig. 3.20) ;

A7, pour afficher la confirmation de réservation (voir Fig. 3.21) ;

A8, pour annuler la réservation (voir Fig. 3.22) ;

A9, pour terminer la réservation (voir Fig. 3.23) ;

```
Activity :– tuple(activityId : A4, properties :?,
 services : set(ref(targetApp), ref(Bank), ?),
 invocation : tuple(iName : checkAccountBalance,
  iData : tuple(
   in : tuple(funcCtrlInfo : (sender :?),
    funcInput : list(clientRef :?, amount :?)),
   out : tuple(funcCtrlInfo : (sender :?),
    funcOutput : list(autorisation :?))),
   iControl : (communication : tuple(
    proxies : ref(Bank),
    bindings : tuple(protocols : ref(invoke),
    dataPackets : ϕ)))),
 preconditions :?, postconditions :?
)
```

FIG. 3.18 – Activité A4

```
Activity :– tuple(activityId : A5, properties :?,
 services : set(ref(targetApp), ref(Bank), ?),
 invocation : tuple(iName : debit,
  iData : tuple(
   in : tuple(funcCtrlInfo : (sender :?),
    funcInput : list(clientRef :?, amount :?)),
   out : tuple(funcCtrlInfo : (sender :?),
    funcOutput : list(paymentRef :?))),
   iControl : (communication : tuple(
    proxies : ref(Bank),
    bindings : tuple(protocols : ref(invoke),
    dataPackets : ϕ)))),
 preconditions :?, postconditions :?
)
```

FIG. 3.19 – Activité A5

```
Activity :– tuple(activityId : A6, properties :?,
 services : set(ref(targetApp), ref(Bank), ?),
 invocation : tuple(iName : credit,
 iRef : Bank,
 iContext : tuple(proxies : ref(Bank),
   protocols : ref(invoke), dataPackets : ϕ),
 iData : tuple(
  in : tuple(funcCtrlInfo : (sender :?),
   funcInput : list(clientRef :?, amount :?)),
  out : tuple(funcCtrlInfo : (sender :?),
   funcOutput : list(paymentRef :?))),
  iControl : (communication : tuple(
   proxies : ref(Bank),
   bindings : tuple(protocols : ref(invoke),
   dataPackets : ϕ)))),
 preconditions :?, postconditions :?
)
```

FIG. 3.20 – Activité A6

```
Activity :– tuple(activityId : A7, properties :?,
 services : set(ref(targetApp), ref(Agence), ?),
 invocation : tuple(iName : displayResult,
  iData : tuple(
   in : tuple(funcCtrlInfo : (sender :?),
    funcInput : list(paymentRef :?, bookingRef :?)),
   out : tuple(funcCtrlInfo : (sender :?),
    funcOutput : list(confirmation :?))),
   iControl : (communication : tuple(
    proxies : ref(Agence),
    bindings : tuple(protocols : ref(invoke),
    dataPackets : ϕ)))),
 preconditions :?, postconditions :?
)
```

FIG. 3.21 – Activité A7

```
Activity :– tuple(activityId : A8, properties :?,
 services : set(ref(targetApp), ref(Agence), ?),
 invocation : tuple(iName : cancelReservation,
  iData : tuple(
   in : tuple(funcCtrlInfo : (sender :?),
    funcInput : list(clientRef :?, bookingRef :?)),
   out : tuple(funcCtrlInfo : (sender :?),
    funcOutput : list(confirmation :?))),
  iControl : (communication : tuple(
   proxies : ref(Agence),
   bindings : tuple(protocols : ref(invoke),
   dataPackets : ϕ)))),
 preconditions :?, postconditions :?
)
```

FIG. 3.22 – Activité A8

```
Activity :– tuple(activityId : A9, properties :?,
 services : set(ref(targetApp), ?),
 invocation : tuple(iName : close,
  iData : tuple(
   in : tuple(funcCtrlInfo : (sender :?), funcInput : ϕ),
   out : tuple(funcCtrlInfo : (sender :?), funcOutput : ϕ)),
  iControl : (communication : tuple(
   proxies : targetApp,
   bindings : tuple(protocols : ref(invoke),
   dataPackets : ϕ)))),
 preconditions :?, postconditions :?
)
```

FIG. 3.23 – Activité A9

3.4.2d Spécification des contraintes des aspects coordination

La contrainte globale de coordination régissant les aspects coordination du plan est définie par

$$\textit{coordinationPolicy} : (\texttt{CoordPol} \rightarrow (\texttt{CoordPol1} \vee \texttt{CoordPol2} \vee \texttt{CoordPol3}))$$

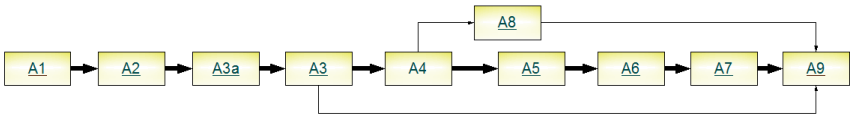
La formule CoordPol décrit le choix entre les trois contraintes globales de coordination permettant de régir une session de réservation selon la logique d'applicative décrite ci-dessous. Elle est étiquetée comme *coordinationPolicy* ; elle est donc associée à toute activité du plan comme leur propriété *coordinationPolicy* [12].

Les contraintes spécialisées constituant cette contrainte globale sont d'un des trois aspects : ordonnancement, déclenchement et inter-dépendance de données. Chaque contrainte spécialisée est une instance d'un prédicat prédéfini d'un des ces aspects. Elle est associée aux activités en tant que pré-condition ou post-condition en respectant les quatre premières règles du guide d'association (voir la section 3.4.1).

La figure 3.24 illustre les contraintes globales spécifiant une session de réservation complète : le client choisit le vol, fait la réservation et règle le billet. Ces contraintes doivent être conjointement satisfaites. Dans ce cas, la contrainte globale des aspects coordination correspond à la formule bien formée :

$$\texttt{CoordPol1} \rightarrow (\texttt{OGC}_1 \wedge \texttt{OGC}_2 \wedge \texttt{FGC}_1 \wedge \texttt{DGC}_1 \wedge \texttt{DGC}_2)$$

où :



$$\texttt{OGC}_1 \rightarrow (\texttt{after}(\mathsf{A2}, \mathsf{A1}) \wedge \mathsf{after}(\mathsf{A3a}, \mathsf{A2}) \wedge \mathsf{after}(\mathsf{A3}, \mathsf{A3a}))$$
$$\texttt{OGC}_2 \rightarrow (\texttt{after}(\mathsf{A4}, \mathsf{A3}) \wedge \mathsf{after}(\mathsf{A5}, \mathsf{A4}) \wedge \mathsf{after}(\mathsf{A6}, \mathsf{A5}) \wedge \mathsf{after}(\mathsf{A7}, \mathsf{A6}) \wedge \mathsf{after}(\mathsf{A9}, \mathsf{A7}))$$
$$\texttt{FGC}_1 \rightarrow (\texttt{forces}(\mathsf{A}(\mathit{State}()), \mathit{State}(\texttt{COMMIT})) \wedge (\forall \texttt{Activity}(\mathsf{A}) \wedge \neq (\mathsf{A}, \mathsf{A8})))$$
$$\begin{aligned}\texttt{DGC}_1 \rightarrow (&\texttt{matches}(\mathsf{A2}(\mathit{input}(\mathit{fullflightdesc}())), \mathsf{A1}(\mathit{output}(\mathit{availableflightlist}())))\\ &\wedge \texttt{matches}(\mathsf{A3a}(\mathit{input}(\mathit{chosenflightdesc}())), \mathsf{A2}(\mathit{output}(\mathit{chosenflightdesc}())))\\ &\wedge \texttt{matches}(\mathsf{A3}(\mathit{input}(\mathit{chosenflightdesc}())), \mathsf{A2}(\mathit{output}(\mathit{chosenflightdesc}())))\\ &\wedge \texttt{matches}(\mathsf{A3}(\mathit{input}(\mathit{clientRef}())), \mathsf{A3}(\mathit{output}(\mathit{clientRef}()))))\end{aligned}$$
$$\begin{aligned}\texttt{DGC}_2 \rightarrow (&\texttt{belongs}(\mathsf{A4}(\mathit{input}(\mathit{amount}())), \mathsf{A3}(\mathit{output}(\mathit{bookingRef}())))\\ &\wedge \texttt{belongs}(\mathsf{A5}(\mathit{input}(\mathit{amount}())), \mathsf{A3}(\mathit{output}(\mathit{bookingRef}())))\\ &\wedge \texttt{belongs}(\mathsf{A6}(\mathit{input}(\mathit{amount}())), \mathsf{A3}(\mathit{output}(\mathit{bookingRef}())))\\ &\wedge \texttt{matches}(\mathsf{A7}(\mathit{input}(\mathit{bookingRef}())), \mathsf{A3}(\mathit{output}(\mathit{bookingRef}())))\\ &\wedge \texttt{matches}(\mathsf{A7}(\mathit{input}(\mathit{paymentRef}())), \mathsf{A5}(\mathit{output}(\mathit{paymentRef}()))))\end{aligned}$$

FIG. 3.24 – Contraintes globales pour une session de réservation complète

- les contraintes globales OGC_1, OGC_2 portent sur l'aspect ordonnancement. OGC_1 spécifie l'ordre séquentiel (illustrée par les flèches grasses) des activités de A1 à A3. OGC_2 spécifie l'ordre séquentiel des activités de A4 à A7.
- la contrainte globale FGC_1 porte sur l'aspect déclenchement. Il spécifie que toute activité du plan, sauf A8, doit se terminer à l'état COMMIT.
- les contraintes globales DGC_1, DGC_2 portent sur l'aspect inter-dépendance de données.

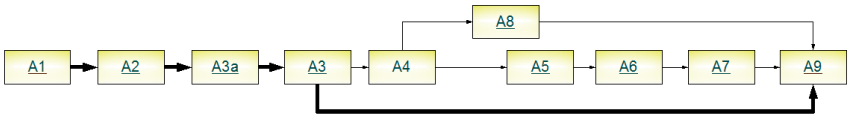
[12] Property :– tuple(*propertyName* : coordinationPolicy, *propertyValue* : Formula) ∈ dom(SafetyPrprt)

DGC_1 spécifie le flux de données entre les activités de A1 à A3. DGC_2 spécifie le flux de données entre les activités de A3 à A7.

La figure 3.25 illustre les contraintes globales spécifiant une session de réservation incomplète : le client choisit le vol et fait la réservation. Ces contraintes doivent être conjointement satisfaites. Dans ce cas, la contrainte globale des aspects coordination correspond à la formule bien formée :

$$\texttt{CoordPol2} \rightarrow (\texttt{OGC}_1 \wedge \texttt{OGC}_3 \wedge \texttt{FGC}_2 \wedge \texttt{DGC}_1)$$

La contrainte globale OGC_3 portant sur l'aspect ordonnancement spécifie l'ordre séquentiel des activités A3, A4 et A9.



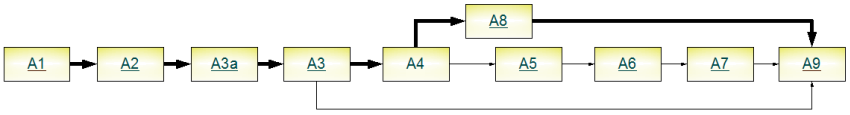
$\texttt{OGC}_3 \rightarrow (\texttt{after}(\texttt{A4}, \texttt{A3}) \wedge \texttt{after}(\texttt{A9}, \texttt{A3}))$
$\texttt{FGC}_2 \rightarrow (\texttt{forces}(\texttt{A}(\textit{State}()), \textit{State}(\texttt{COMMIT})) \wedge (\forall \texttt{Activity}(\texttt{A}) \wedge \notin (\texttt{A}, \texttt{set}(\texttt{A4}, \texttt{A5}, \texttt{A6}, \texttt{A7}, \texttt{A8}))))$

FIG. 3.25 – Contraintes globales pour une session de réservation incomplète

La figure 3.26 illustre une contrainte globale spécifiant une session de réservation non-autorisée : le client choisit le vol et fait la réservation ; cette réservation est automatiquement annulée à cause du solde insuffisant de compte bancaire du client. Dans ce cas, la contrainte globale des aspects coordination correspond à la formule bien formée :

$$\texttt{CoordPol3} \rightarrow (\texttt{OGC}_1 \wedge \texttt{OGC}_4 \wedge \texttt{FGC}_3 \wedge \texttt{DGC}_3)$$

La contrainte globale OGC_4 portant sur l'aspect ordonnancement spécifie l'ordre séquentiel des activités A3, A4, A8 et A9. La contrainte globale FGC_3 portant sur l'aspect déclenchement spécifie l'exécution avec succès des activités de A1 à A4 et A8. La contrainte globale DGC_3 portant sur l'aspect inter-dépendance de données spécifie le flux de données entre les activités A3, A4 et A8.



$\texttt{OGC}_4 \rightarrow (\texttt{after}(\texttt{A4}, \texttt{A3}) \wedge \texttt{after}(\texttt{A8}, \texttt{A4}) \wedge \texttt{after}(\texttt{A9}, \texttt{A8}))$
$\texttt{FGC}_3 \rightarrow (\texttt{forces}(\texttt{A}(\textit{State}()), \textit{State}(\texttt{COMMIT})) \wedge (\forall \texttt{Activity}(\texttt{A}) \wedge \notin (\texttt{A}, \texttt{set}(\texttt{A5}, \texttt{A6}, \texttt{A7}))))$
$\texttt{DGC}_3 \rightarrow (\texttt{matches}(\texttt{A8}(\textit{input}(\textit{clientRef}()))), \texttt{A3}(\textit{input}(\textit{clientRef}()))) \wedge$
$\texttt{matches}(\texttt{A8}(\textit{input}(\textit{bookingRef}()))), \texttt{A3}(\textit{output}(\textit{bookingRef}())))$

FIG. 3.26 – Contraintes globales pour une session de réservation non-autorisée

3.4.2e Spécification des contraintes des aspects sécurité

Certaines contraintes globales de sécurité sont définies pour régir les aspects sécurité de l'exécution du plan.

Authentification La figure 3.27 représente une politique d'authentification qui est définie par la formule bien formée authenPol. Cette formule correspond à la conjonction de deux contraintes globales suivantes :

- authenPol1 permet d'authentifier les activités A4, A5, A6 ;
- authenPol2 permet d'authentifier les services appelant et appelé dans leur cadre.

$$\begin{aligned}&\textit{authenticationPolicy} : (\mathsf{authenPol} \rightarrow (\mathsf{authenPol1} \wedge \mathsf{authenPol2}))\\&\mathsf{authenPol1} \rightarrow (\mathsf{authentic}(\mathsf{A4}, \mathsf{ref}(\mathsf{algo1})) \wedge \mathsf{authentic}(\mathsf{A5}, \mathsf{ref}(\mathsf{algo1})) \wedge \mathsf{authentic}(\mathsf{A6}, \mathsf{ref}(\mathsf{algo2})))\\&\mathsf{authenPol2} \rightarrow (\mathsf{authentic}(\mathsf{a}(\textit{services}()), \mathsf{ref}(\mathsf{algo3})) \wedge (\forall \mathsf{a} \in \mathsf{set}(\mathsf{A4}, \mathsf{A5}, \mathsf{A6})))\end{aligned}$$

FIG. 3.27 – Contrainte globale d'authentification

La contrainte globale authenPol définissant cette politique d'authentification est étiquetée comme *authenticationPolicy* ; elle est donc associée aux activités A4, A5, A6 en tant que propriété *authentication Policy*[13].

Les contraintes spécialisées constituant la contrainte globale authenPol sont des instances du prédicat `authentic`() de l'aspect authentification. Elles sont respectivement associées aux activités A4, A5, A6 en tant que pré-condition de leurs appels.

Autorisation La figure 3.28 représente une politique d'autorisation qui est définie par une formule bien formée. Elle permet de vérifier la permission d'appel dans le cadre de chaque activité du plan (référencée par le nom a). Cette formule est étiquetée comme *authorisationPolicy* ; elle est donc associée à toute activité du plan comme leur propriété *authorisationPolicy* [14].

$$\begin{aligned}&\textit{authorisationPolicy} :\\&\quad(\mathtt{obligates}(\mathsf{a}(\mathsf{Invocation}(\textit{iName}(\mathsf{f}), \textit{iControl}(\textit{AuthorisationPrprt}(\textit{Certificate}()))))),\\&\qquad\qquad\mathtt{Service}(\mathtt{Operation}(\textit{operationName}(\mathtt{f}))))\\&\quad\wedge \forall \mathtt{Activity}(\mathsf{ref}(\mathsf{a})))\end{aligned}$$

FIG. 3.28 – Contrainte globale d'autorisation

Les contraintes spécialisées constituant cette contrainte globale d'autorisation sont des instances de la formule `obligate`() $\wedge$ `Activity`(). Cette formule est construite de manière non récursive à partir du prédicat `obligate`() de l'aspect autorisation. Ces contraintes spécialisées sont respectivement associées aux activités du plan en tant que pré-condition de leurs appels.

Intégrité La figure 3.29 représente une politique d'intégrité qui est définie par la formule bien formée integPol. Cette formule correspond à la conjonction de deux contraintes globales suivantes :

- integPol1 permet d'assurer l'intégrité des messages échangés entre le couple d'activités A3a et A3 et entre le couple d'activité A3 et A4 ;
- integPol2 permet d'assurer l'intégrité des messages échangés entre les services appelant et appelé correspondant.

Cette formule est étiquetée comme *integrityPolicy* ; elle est donc associée aux activités A3a, A3, A4 en tant que propriété *integrityPolicy* [15]. Les contraintes spécialisées constituant cette contrainte globale d'intégrité sont des instances du prédicat `unmodified`() de l'aspect intégrité. Elles sont associées aux activités conformément aux règles 1 et 7 du guide d'association (voir la section 3.4.1).

[13] Property :– tuple(*propertyName* : authenticationPolicy, *propertyValue* : Formula) $\in$ dom(SafetyPrprt)
[14] Property :– tuple(*propertyName* : authorisationPolicy, *propertyValue* : Formula) $\in$ dom(SafetyPrprt)
[15] Property :– tuple(*propertyName* : integrityPolicy, *propertyValue* : Formula) $\in$ dom(SafetyPrprt)

integrityPolicy : (integPol → (integPol1 ∧ integPol2))
integPol1 → (
unmodified(A3(Invocation(*iData*(), *iControl*(*recvmsg*()))), A3a(Invocation(*iData*(), *iControl*(*sentmsg*())))∧
unmodified(A4(Invocation(*iData*(), *iControl*(*recvmsg*()))), A3(Invocation(*iData*(), *iControl*(*sentmsg*()))))
)
integPol2 → (
unmodified(a(Invocation(*iName*(f), *iData*())), Service(f(*input*(), *output*()))∧
(∀a ∈ set(A3a, A3, A4)) ∧ Operation(ref(f)))

FIG. 3.29 – Contrainte globale d'intégrité

Non-répudiation La figure 3.30 représente une politique de non-répudiation permettant de vérifier l'ordre de l'exécution des activités A1..A7 avant d'exécuter l'activité $A8$. La contrainte globale correspondant à cette politique de non-répudiation est définie par une formule bien formée. Cette formule est étiquetée comme *nonrepudiationPolicy* ; elle est donc associée à toute activité du plan comme sa propriété *nonrepudiationPolicy* [16].

nonrepudiationPolicy : (ordered(Activity(*activityId*(A7))))

FIG. 3.30 – Contrainte globale de non-répudiation

Cette contrainte globale de non-répudiation correspond à une instance du prédicat `ordered`() de l'aspect non-répudiation. Elle représente à la fois sa contrainte spécialisée constituante. Conformément aux règles 1 et 8 du guide d'association (voir la section 3.4.1), elle peut être associée à l'activité A7 en tant que post-condition. Elle peut être également associée à l'activité A8 en tant que pré-condition.

La figure 3.31 illustre les contraintes spécialisées de différents aspects qui sont associées à l'activité A4 en tant que ses pré et post conditions.

```
Activity :– tuple(activityId : A4,
  properties : ■,
  services : ■, invocation : ■,
  preconditions : set(
    after(A4, A3),
    belongs(A4(input(amount())), A3(output(bookingRef()))),
    authentic(A4, ref(algo1)),
    authentic(A4(services()), ref(algo3)),
    obligates(A4(Invocation(iName(checkAccountBalance), iControl(AuthorisationPrprt(Certificate()))),
              Service(Operation(operationName(checkAccountBalance)))),
    unmodified(A4(Invocation(iData(), iControl(recvmsg()))), A3(Invocation(iData(), iControl(sentmsg())))),
    unmodified(A4(Invocation(iName(checkAccountBalance), iData())),
              Service(checkAccountBalance(input(), output()))
  ),
  postconditions : set(
    forces(A4(State()), State(COMMIT))

  ))
```

FIG. 3.31 – Spécification des préconditions et post conditions de l'activité A4

3.4.2f Spécification des propriétés

Des propriétés des aspects coordination et de sécurité sont déclarées à fournir (voir FIG. 3.32). Les propriétés servant à la génération du plan et leur évaluation sont aussi fournies.

[16] Property :– tuple(*propertyName* : nonrepudiationPolicy, *propertyValue* : Formula) ∈ dom(SafetyPrprt)

```
Activity :– tuple(activityId : ■,
    properties : set(
        requiredProperties : φ,
        providedProperties : set(
            orderingPrprts : set(timing, timingTrace),
            firingPrprts : set(eState, stateTrace),
            dataPrprts : set(in, out, dataTrace),
            AuthenSubjProof : Certificate,
            AuthenRule : Operation,
            AuthorSubjProof : Certificate,
            safetyPrprts : set(coordinationPolicy, authenticationPolicy, autorisationPolicy,
                               integrityPolicy, nonrepudiationPolicy),
            ?
        )
    ),
    services : ■, invocation : ■,
    preconditions :?, postconditions :?
)
```

Fig. 3.32 – Propriétés prévues à fournir par des activités

La spécification du plan de coordination sécurisée d'activités de l'application de réservation de vols se ramène donc à la spécification des activités A1..A9. La spécification complète de chaque activité appartenant du plan contient les informations suivantes :

- des propriétés nécessaires pour l'exécution de ce plan ;
- des services servant à son exécution dans ce plan ;
- des contraintes qui doivent être satisfaites avant et/ou après l'appel ;
- des preuves d'exécution qui seront archivées dans des journaux de coordination lors de l'exécution de ce plan.

Conclusion

Le modèle MEO offre des concepts "propriété", "service", "contrainte", "activité" et "journal de coordination" pour spécifier de différents aspects de l'exécution coordonnée des services. Avec le modèle, chaque aspect considéré peut-être décrit comme des contraintes qui sont définies en utilisant les prédicats et propriétés. Le modèle offre certaines propriétés et prédicats propres aux aspects que nous considérons pour la coordination sécurisée des activités. Le modèle est ouvert et permet ainsi par la définition de nouveaux prédicats et propriétés de prendre des aspects non encore considérés. Il faut noter également que le modèle permet ainsi de prendre en compte d'éventuels nouveaux besoins apparaissant sur chaque aspect (nous pouvons ajouter, supprimer ou redéfinir les propriétés et prédicats que nous avons pré-définis pour chaque aspect). MEO permet d'appréhender d'une façon globale, cohérente et uniforme les aspects considérés au travers de la notion centrale de contraintes. L'évaluation des contraintes représente le moteur de l'exécution coordonnée des services (l'exécution de la coordination sécurisée des activités d'une application à base de services). Le chapitre 4 décrit une sémantique de cette évaluation. L'approche MEO introduit la capacité de réaliser l'évaluation des contraintes globales de différents aspects de façon homogène. Elle est complètement matérialisée par le canevas logiciel MEOBI qui offre cette flexibilité dans une infrastructure d'exécution de la coordination sécurisée. Ce canevas est présenté au chapitre 5.

Chapitre 4

Exécution de la coordination sécurisée

Ce chapitre décrit la manière dont est exécuté un plan de coordination sécurisée d'activités. L'exécution d'un tel plan consiste globalement à exécuter ses activités. Le point de départ de l'exécution du plan est l'exécution de(s) activité(s) qui ne possède(nt) pas de prédécesseur. L'exécution d'un plan se termine par l'exécution de(s) activité(s) qui ne possède(nt) pas de successeur. L'exécution réussie d'une activité déclenche automatiquement l'exécution de son(ses) successeur(s). Le cas échéant, l'exécution de toutes les activités qui succèdent à une activité défaillante n'a pas lieu. Les propriétés des activités (définies dans le plan ou générées à partir du plan) fournissent les informations utiles et nécessaires pour l'exécution du plan. Comme illustrée dans la figure 4.1, cette exécution requiert dans un premier temps la transformation du plan spécifié en un plan exécutable. Cela est fait en ajoutant aux activités du plan spécifié certaines propriétés servant à son exécution. Les activités spécifiées dans le plan exécutable correspondent aux systèmes de contraintes (spécialisées, locales, globales, d'appel) devant être évaluées avec ces propriétés. Dans un deuxième temps, l'exécution du plan consiste en l'évaluation de ses contraintes en utilisant ces propriétés. Elle utilise des évaluateurs de contraintes. La sûreté de fonctionnement de l'exécution du plan est assurée à la fois par (i) la satisfaction de contraintes (notamment de sécurité), et (ii) le traitement d'exception lors de l'occurrence d'une contrainte non satisfaite.
Ce chapitre est organisé de manière suivante. La section 4.1 introduit le processus d'exécution d'une activité. La section 4.2 présente la transformation du plan spécifié en un plan exécutable en détaillant les informations utiles et nécessaires pour cette exécution. Ces informations sont soit définies dans le plan, soit générées à partir du plan. La section 4.3 présente l'approche et l'algorithme général d'évaluation pour des formules représentant les contraintes du plan. Elle présente aussi l'algorithme de traitement d'exception. Les sections 4.4 à 4.6 précisent l'évaluation des contraintes spécialisées, locales et globales du plan.

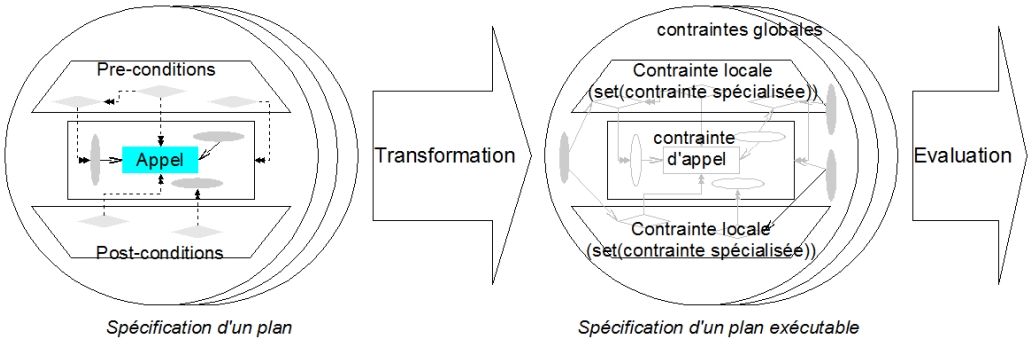


Fig. 4.1 – Exécution d'un plan de coordination sécurisée d'activités

De plus, nous étudions comment l'exécution d'un plan peut s'adapter aux nouveaux besoins et aux changements (de propriétés, de contraintes, d'évaluateurs, etc.) intervenant pendant l'exécution elle-même. Une telle adaptation s'exprime en termes de propriétés et de contraintes spécifiques au plan. Ces propriétés et contraintes, quant à elles, sont définies dans le plan, générées à partir du plan ou renseignées lors de l'exécution du plan. De cette façon, l'adaptation peut être respectivement autorisée au moment de la définition du plan, de l'interprétation du plan ou au cours de l'exécution du plan. Cette adaptation est gérée de manière similaire à d'autres aspects considérés du plan. Tout cela est abordé dans la section 4.7.

4.1 Exécution d'une activité

La figure 4.2 schématise l'exécution d'une activité dont la pré-condition de l'appel, l'appel lui-même et sa post-condition sont traités de manière intéractive et configurable. Comme nous

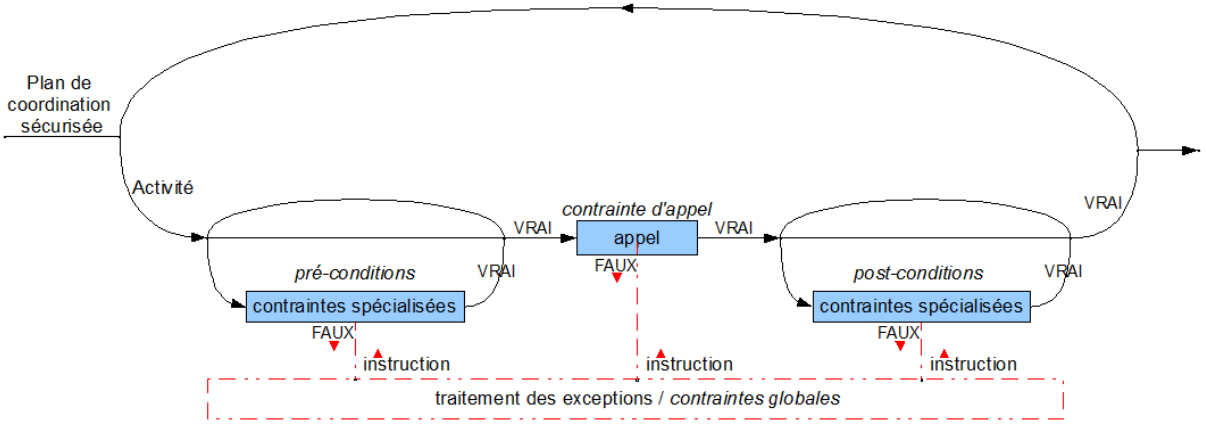


FIG. 4.2 – Exécution d'une activité d'un plan de coordination sécurisée

l'avons déjà dit, l'exécution du plan de coordination sécurisée d'activités consiste donc à évaluer des contraintes (de coordination, de sécurité, etc.) du plan. Elle consiste également à évaluer si les appels aux fonctions de différents services se sont bien passés.

L'exécution d'une activité est conforme aux règles données dans la figure 4.3. Ces règles définissent à la fois les processus d'exécution et les états d'exécution d'une activité.

$$activity-execution ::= treatment \mid (preparation \wedge treatment \wedge termination) ; \quad (4.1.1)$$

$$preparation ::= pre-checking \mid (pre-checking \wedge exception-treatment) \mid UNDER\_CONSIDERATION \mid WAITING \mid READY ; \quad (4.1.2)$$

$$treatment ::= function-invocation \mid (function-invocation \wedge exception-treatment) \mid ACTIVE ; \quad (4.1.3)$$

$$termination ::= post-checking \mid (post-checking \wedge exception-treatment) \mid ABORT \mid COMMIT ; \quad (4.1.4)$$

$$pre-checking ::= constraint-evaluation \mid VRAI \mid FAUX ; \quad (4.1.5)$$

$$post-checking ::= constraint-evaluation \mid VRAI \mid FAUX ; \quad (4.1.6)$$

$$exception-treatment ::= PAUSE \mid COMPENSATIVE ; \quad (4.1.7)$$

FIG. 4.3 – Règles de contrôle d'exécution pour une activité

Conformément à la règle 4.1.1, le processus d'exécution d'une activité est composé de trois

phases : *preparation*, *treatment* et *termination*.

- *preparation* : cette phase consiste à évaluer les contraintes spécialisées qui sont associées à cette activité en tant que pré-condition de son appel ;
- *treatment* : cette phase consiste à évaluer l'appel ; et
- *termination* : cette phase consiste à évaluer les contraintes spécialisées qui sont associées à cette activité en tant que post-condition de son appel.

La correspondance entre la spécification d'une activité et son processus d'exécution est illustrée dans la figure 4.4. Les phases de préparation et de terminaison sont optionnelles, s'il n'y a aucune contrainte devant être satisfaite avant et/ou après l'appel.

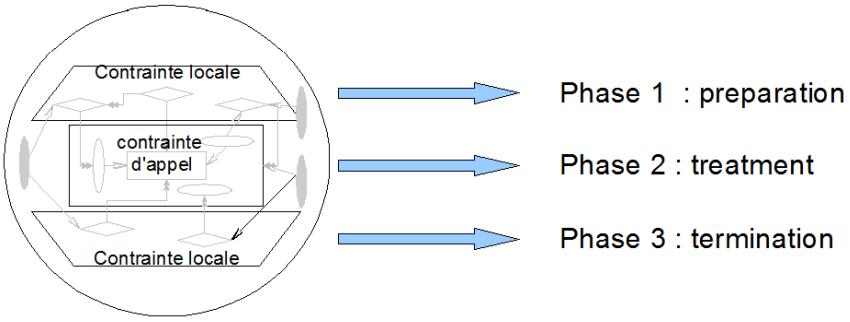


FIG. 4.4 – Phases de l'exécution d'une activité

La figure 4.5 illustre l'automate états-transitions d'une activité durant son exécution. Cet automate ainsi que les autres règles de la figure 4.3 sont détaillés ci-dessous lors de la description des phases correspondantes du processus d'exécution.

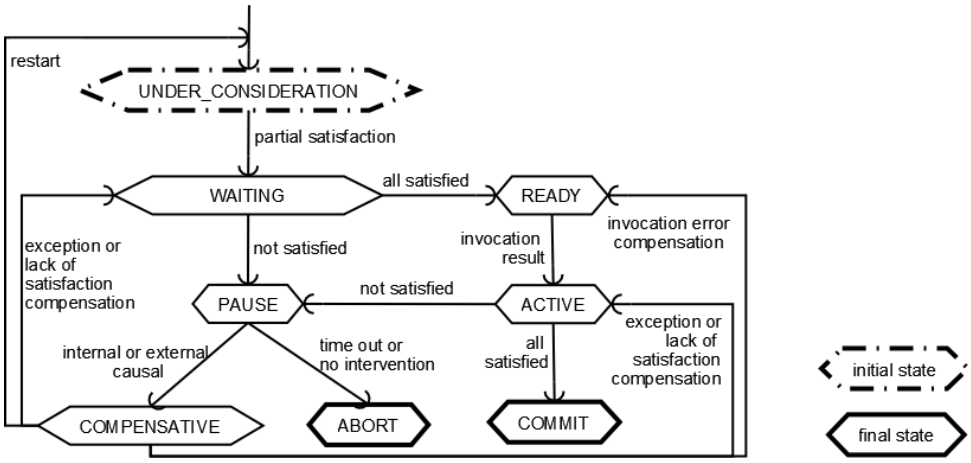


FIG. 4.5 – Automate états-transitions d'exécution d'une activité

4.1.1 Phase "Preparation"

Conformément aux règles 4.1.2 et 4.1.7, la phase de préparation du processus d'exécution d'une activité correspond à l'évaluation des contraintes spécialisées ; ces dernières sont associées à l'activité et doivent être satisfaites avant l'exécution de l'appel (pré-condition). Du point de vue local, la pré-condition d'un appel porte sur les aspects déclenchement, authentification, autorisation. Du point de vue global, la pré-condition d'un appel porte sur les aspects ordonnancement, inter-dépendance de données et intégrité. Elle porte éventuellement sur les aspects adaptation ainsi que sur d'autres aspects de contrôle.

L'évaluation des contraintes spécialisées de la pré-condition utilise (i) les informations extraites de la spécification de l'activité considérée et (ii) les preuves d'exécution des activités concernées comme les paramètres d'entrée. Elle renvoie un résultat booléen qui indique si les contraintes spécialisées correspondantes sont satisfaites ou non.

Cette phase concerne les états UNDER_CONSIDERATION, WAITING, READY, PAUSE de l'automate de la figure 4.5. UNDER_CONSIDERATION est l'état initial de l'exécution d'une activité. Cet état devient WAITING une fois que le processus d'évaluation des contraintes spécialisées de la pré-condition d'appel est démarré. L'état reste WAITING jusqu'au moment où les contraintes spécialisées ont toutes été évaluées. Si toutes ces contraintes sont satisfaites alors l'état d'exécution de l'activité devient READY. Si une contrainte n'est pas satisfaite, l'état d'exécution de l'activité devient PAUSE.

4.1.2 Phase "Treatment"

Conformément aux règles 4.1.2 et 4.1.3, la phase de traitement correspond à l'évaluation de l'appel d'une fonction d'un service à coordonner. Cette évaluation utilise les informations sur l'exécution de l'appel, et les informations extraites de la spécification de l'activité considérée. L'appel est évalué après que son résultat ait été obtenu. Le résultat d'appel est soit renvoyé par le service appelé (service à coordonner), soit renvoyé par le service appelant. Le résultat de l'évaluation est un booléen. L'évaluation renvoie *FAUX* en cas d'exécution défaillante de l'appel.

Cette phase concerne les états READY, ACTIVE, PAUSE de l'automate de la figure 4.5. Dans l'état READY, l'appel à une fonction d'un service à coordonner est exécuté. Une fois que le résultat de l'appel est retourné, l'état d'exécution est ACTIVE. Si le résultat d'appel montre que l'exécution d'appel s'est bien passée, l'état de l'activité reste ACTIVE. Sinon, cet état devient PAUSE.

4.1.3 Phase "Termination"

Conformément aux règles 4.1.2 et 4.1.4, la phase de terminaison correspond à l'évaluation des contraintes spécialisées ; ces dernières sont associées à l'activité et doivent être satisfaites après l'exécution réussie de l'appel (post-condition). Du point de vue local, la post-condition porte sur les aspects inter-dépendance de données et non-répudiation. Du point de vue global, la post-condition porte sur les aspects ordonnancement, déclenchement et intégrité. Ces post-conditions portent éventuellement sur les aspects adaptation ainsi que sur d'autres aspects contrôle, par exemple le traitement lors de la non satisfaction d'une contrainte.

L'évaluation des contraintes spécialisées de la post-condition d'un appel utilise (i) les informations extraites de la spécification de l'activité considérée et (ii) les preuves d'exécution des activités concernées comme les paramètres d'entrée. Elle renvoie un résultat booléen qui indique si les contraintes spécialisées sont satisfaites ou non.

Cette phase concerne les états ACTIVE, COMMIT, ABORT, PAUSE, COMPENSATIVE de l'automate donnée dans la figure 4.5. Dans l'état ACTIVE, le processus d'évaluation des contraintes spécialisées de la post-condition d'appel est démarré. Si toutes ces contraintes sont satisfaites, l'exécution de l'activité se termine dans l'état final COMMIT. Si une contrainte n'est pas satisfaite, l'état d'exécution de l'activité devient PAUSE.

Dans l'état PAUSE, le processus d'exécution est en attente d'une intervention. Il s'agit d'une intervention interactive (par exemple l'intervention causée par l'utilisateur ou par un service tiers de confiance), configurable (par exemple l'intervention en fonction de la transformation

des propriétés d'exécution) ou automatique (par exemple l'intervention interne ou par défaut causée par le moteur d'exécution du plan). En fonction de l'intervention, l'état d'exécution est passé à COMPENSATIVE, puis à WAITING (pour compenser l'évaluation d'une pré-condition non satisfaite ou ne pas prendre en compte la satisfaction de cette contrainte), ou bien à READY (pour compenser l'exécution de l'appel), ou bien à ACTIVE (pour reprendre l'évaluation d'une post-condition non satisfaite ou ne pas prendre en compte la satisfaction de cette contrainte ou ne pas prendre en compte le succès de l'exécution de l'appel), ou bien à UNDER_CONSIDERATION (pour réinitialiser l'exécution du processus). Le cas échéant (pas d'intervention ou dépassement du délai d'attente), l'exécution de l'activité se termine dans l'état final ABORT. Cet état signifie l'annulation de l'exécution de l'activité courante.

4.1.4 Enchaînement

L'enchaînement des phases d'un processus d'exécution d'une activité, ainsi que l'enchaînement des processus d'exécution des activités, se basent sur le résultat de l'évaluation des contraintes dans chaque phase.

- Si le résultat d'évaluation de toutes les contraintes spécialisées de la pré-condition de l'appel est *VRAI* (toutes ces contraintes sont satisfaites), alors l'évaluation de l'exécution de l'appel est déclenchée.
- Si le résultat d'évaluation de l'exécution de l'appel est *VRAI* (l'exécution de l'appel s'est bien passée), alors l'évaluation des contraintes spécialisées de la post-condition de l'appel est déclenchée.
- Si le résultat d'évaluation de toutes les contraintes spécialisées de la post-condition de l'appel est *VRAI* (toutes ces contraintes sont satisfaites), alors l'exécution de(s) activité(s) successive(s) est déclenchée.
- Si le résultat d'évaluation de contrainte(s) est *FAUX* (contrainte(s) non satisfaite(s)), cela est considéré comme une exception. Le traitement d'une exception permet d'adapter la suite de l'exécution du processus : réinitialiser le processus, reprendre le processus à l'état d'exécution avant le moment où l'exception est survenue, ou continuer le processus sans prendre en compte l'exception. Par défaut, une exception annule l'exécution de l'activité courante.

4.1.5 Instanciation des paramètres

Des données sont échangées entre des activités au sein de l'évaluation de leurs contraintes d'inter-dépendance de données. Cet échange correspond à l'affectation des valeurs aux paramètres de ces contraintes au moment de l'évaluation.

Des données (d'entrée et de sortie de l'exécution de la fonction appelée) sont échangées entre des services appelant et appelés au sein de l'évaluation de l'appel correspondant à chaque activité du plan. Cet appel correspond à l'affectation des valeurs aux paramètres de contrainte de l'appel au moment de l'évaluation.

4.1.6 Preuves d'exécution

Des preuves d'exécution sont archivées à la fin de chaque évaluation conformément à la spécification des activités : seules les preuves d'exécution qui sont précisées dès la spécification sont archivées.

4.1.7 Exemple

Considérons l'application de réservation de vols que nous avons spécifiée dans la section 3.4.2. Le plan de coordination sécurisée correspondant est spécifié par un ensemble d'activités A1..A9 (voir la section 3.4.2c). L'exécution de chaque activité du plan est conforme aux principes d'exécution décrits dans la section 4.1 et à la spécification de l'activité.

La figure 4.6 illustre l'enchaînement des activités de ce plan. Cet enchaînement est déterminé en fonction des propriétés spécifiques qui se trouvent dans la spécification des activités. Ces propriétés précisent si une activité possède ou non des prédécesseurs et des successeurs ; elles seront décrites dans la section 4.2. Le point de départ de l'exécution de ce plan est donc l'exécution de l'activité A1 (qui ne possède pas de prédécesseur).

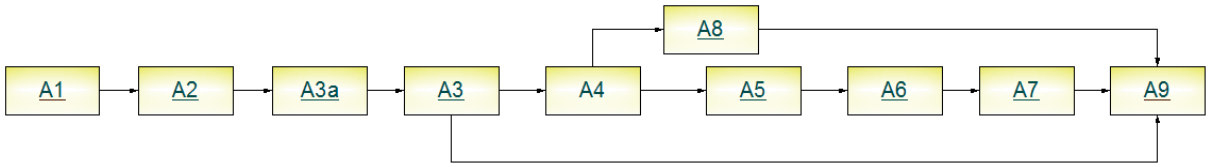


FIG. 4.6 – Enchaînement des activités du plan de coordination sécurisée

Phase "preparation" de l'exécution de l'activité A1 : L'exécution de A1 débute par l'évaluation les contraintes spécialisées qui doivent être satisfaites avant l'exécution de l'appel. Cette évaluation est réalisée par un évaluateur qui implante l'algorithme ALG. 10 (voir la section 4.5). La spécification de l'activité A1 ne précise aucune de ces contraintes, donc la pré-condition de l'appel est satisfaisante.

Phase "treatment" de l'exécution de l'activité A1 : L'évaluation de l'appel est donc déclenchée. Cette évaluation comprend : (i) l'appel à la fonction seekFlight du service *Agence*, et (ii) la vérification si l'exécution de cet appel est réussie. Le résultat d'appel est archivé dans le journal de coordination de A1. Le résultat d'exécution de cet appel est archivé dans le journal de coordination de A1.

Phase "termination" de l'exécution de l'activité A1 : Si l'exécution de cet appel est réussie, la contrainte d'appel est satisfaisante. Cela déclenche l'évaluation de sa post-condition. Cette évaluation est effectuée par un évaluateur implantant l'algorithme ALG. 10.

Dans la spécification de l'activité A1, il y a une seule contrainte spécialisée de déclenchement qui est associée en tant que post-condition : `forces(A1(`*`State`*`()),`*`State`*`(COMMIT))`. L'évaluation de la post-condition revient à l'évaluation de cette contrainte spécialisée de déclenchement. Cette contrainte spécialisée de déclenchement, quant à elle, est évaluée par un évaluateur implantant l'algorithme décrit dans la section 4.4. L'évaluateur utilise une propriété particulière de l'activité pour décider de son fonctionnement ; cette propriété de l'expression de l'évaluation de contrainte (correspondant au critère de regroupement *evalScope*) sera détaillée dans la section 4.2.2. La propriété de l'expression de l'évaluation de contrainte, présente dans la spécification de l'activité A1, précise que la contrainte spécialisée de déclenchement est évaluée sur la portée statique. Dans ce cas, l'évaluateur vérifie si l'exécution de l'activité A1 s'est bien passée en fonction des transitions d'états. Si la contrainte spécialisée de déclenchement est satisfaisante, la post-condition est aussi satisfaisante.

L'exécution de l'activité A2 est donc déclenchée.

Phase "preparation" de l'exécution de l'activité A2 : Les contraintes spécialisées qui sont spécifiées en tant que pré-condition sont évaluées selon l'algorithme Alg. 10. Cela est fait en respectant la priorité d'évaluation de ces contraintes spécialisées. Cette priorité, elle aussi, est définie sous forme d'une propriété particulière de l'activité que l'évaluateur utilise pour déterminer son fonctionnement.

Considérons matches(A2(*input*(*fullflightdesc*())), A1(*output*(*availableflightlist*()))), une contrainte spécialisée d'inter-dépendance de données qui doit être évaluée avant l'appel. Cette contrainte précise que la liste de vols disponibles qui sera utilisée comme le paramètre d'entrée de l'appel dans le cadre de l'activité A2 doit être la liste de vols disponibles produite par l'exécution de l'activité A1. La propriété de l'expression de l'évaluation de contrainte, présente dans la spécification de l'activité A2, précise que cette contrainte d'inter-dépendance de données est évaluée sur la portée dynamique. Dans ce cas, l'évaluation de cette contrainte comprend à la fois l'échange des données entre les activités A1 et A2, et la vérification de l'instanciation correcte des paramètres d'entrée de l'appel dans le cadre de A2.

Phase "termination" de l'exécution de l'activité A4 : Supposons que l'exécution des activités A2 et A3 est réussie et considérons l'exécution de l'activité A4. Si une de ses contraintes (spécialisées, locales, globales) n'est pas satisfaite, le traitement d'exception est déclenché. Si, après le traitement d'exception, la contrainte n'est encore pas satisfaite, l'exécution de l'activité A4 est annulée. L'exécution du plan continue par l'exécution d'une autre activité qui succède à la dernière activité réussie, c.-à-d. l'exécution de l'activité A8. Si l'exécution de A8 est annulée en fonction de la non-satisfaction de ses contraintes, l'exécution du plan s'arrête car il n'y a plus d'activité qui succède à la dernière activité réussie et qui n'est pas déjà exécutée.

4.2 Transformation d'un plan de coordination sécurisée d'activités

Un plan de coordination sécurisée d'activités est transformé en y ajoutant certaines propriétés particulières (c.f. le domaine *renderingPrprt* décrit dans la section 3.3.8). Les noms de ces propriétés sont utilisés pour étiqueter les informations typées spécifiant le plan. Ces étiquettes exhibent les renseignements utiles à l'exécution des activités et à l'évaluation des contraintes, lorsqu'elles sont exploitées pour instancier certains paramètres d'entrée (de l'exécution et de l'évaluation). Par exemple, les étiquettes d'une contrainte permettent d'expliciter quand, où et avec quelle priorité elle est évaluée, dans quel cas est acceptée sa non-satisfaction, etc. La transformation est faite soit manuellement, soit automatiquement dans la phase de transformation (avant de démarrer le processus de l'exécution décrit ci-dessus). L'objectif de cette phase est triple :

- Elle détermine l'enchaînement des activités du plan de coordination et la collection des contraintes à évaluer (en terme des propriétés spécifiques à ajouter à la spécification des activités au moment de la transformation du plan).
- Elle vérifie si les contraintes spécifiées sont bien positionnées, évaluables et cohérentes.
- Elle détermine les possibilités de l'évaluation des contraintes.

4.2.1 Enchaînement des activités d'un plan

La transformation des contraintes spécialisées d'ordonnancement (associées à une activité d'un plan) en propriétés du domaine renderingPrprt détermine l'enchaînement entre cette activité et d'autres activités du plan.

4.2.1a Propriétés

Les propriétés définissant l'enchaînement des activités d'un plan sont :

$$\text{Property} :- \text{tuple}(\textit{propertyName} : \text{predecessors}, \textit{propertyValue} : \text{set}(\text{Activity})) \tag{4.2.1}$$

$$\text{Property} :- \text{tuple}(\textit{propertyName} : \text{successors}, \textit{propertyValue} : \text{set}(\text{Activity})) \tag{4.2.2}$$

La valeur de la propriété de nom *predecessors* d'une activité A d'un plan représente l'ensemble des activités de ce plan qui doivent être exécutées juste avant A.

La valeur de la propriété de nom *successors* d'une activité A d'un plan représente l'ensemble des activités de ce plan qui sont à exécuter juste après A.

Exemple. Considérons l'activité A4, une des activités construisant l'application de réservation de vol décrite dans la section 3.4.2. Cette activité a pour but de vérifier le solde du compte bancaire du client. Conformément à la logique applicative décrite, dans la spécification de cette activité (voir la figure 3.18), les propriétés suivantes sont ajoutées :
Property :− tuple(*propertyName* : predecessors, *propertyValue* : set(ref(A3)))
Property :− tuple(*propertyName* : successors, *propertyValue* : set(ref(A5), ref(A8)))
Nous les écrivons plus simplement :
predecessors : set(ref(A3))
successors : set(ref(A5), ref(A8))
Ces propriétés stipulent que A3 doit être exécutée juste avant A4 et que A5 et A8 doivent être exécutées juste après A4.
Dans le cas de l'exécution réussie de l'activité A4, l'exécution d'une activité, qui est choisie aléatoirement à partir de la valeur ensemble set(ref(A5), ref(A8)) de la propriété successor d'A4, continue l'exécution du plan. Il s'agit de l'activité A4 ou A8.
Dans le cas de l'exécution défaillante de l'activité A4, le point de reprise de l'exécution du plan est déterminé de manière suivante. Tout d'abord, il faut déterminer la dernière activité réussie, c.-à-d. le prédécesseur effectif d'A4. Il s'agit d'une activité qui appartient à la valeur ensemble set(ref(A3)) de la propriété predecessor d'A4 et qui est présente dans le journal de coordination d'A4 en tant que son prédécesseur, c.-à-d. l'activité A3. Ensuite, il faut choisir une activité parmi les successeurs qui ne sont pas exécutés d'A3 pour continuer l'exécution du plan. Cela est fait en utilisant la propriété successors d'A3 comme un de ses paramètres d'entrée.
Dans le cas où ces propriétés ne sont pas spécifiées, elles peuvent être également générées à partir des contraintes d'ordonnancement du plan illustrées dans la figure 3.24. L'algorithme décrit dans la section suivante permet de le faire. Une fois que ces propriétés sont générées, elles sont associées aux activités.

4.2.1b Algorithme de transformation des contraintes d'ordonnancement en propriétés d'enchaînement

[Input : set(prédicat), des prédicats d'ordonnancement]
[Output : retourne un ensemble ordonné des activités, chacune est associée aux prédécesseurs et aux successeurs ; retourne une exception le cas échéant]
makeplan : (*orderingpredicats* : set(Predicate)) → (set(tuple(*consideredActivity* : Activity, *predecessors* : set(choice(Activity, ref(Activity))), *successors* : set(choice(Activity, ref(Activity))))))

- set(activité) ← ϕ
- set(tuple(activité, prédécesseurs, successeurs)) ← ϕ
- **pour chaque** pr ∈ set(prédicat) **faire**
  - **selon que**
    - pr = before(v_1,v_2) : set(activité) ← set(activité) ∪ v_1 ∪ v_2
    - pr = after(v_1,v_2) : set(activité) ← set(activité) ∪ v_1 ∪ v_2
    - pr = simultaneous(v_1,v_2) : set(activité) ← set(activité) ∪ v_1 ∪ v_2
- **pour chaque** activité$_j$ ∈ set(activité) **faire**
  - **pour chaque** prédicat$_i$ ∈ set(prédicat) **faire**
    - **selon que**
      - prédicat$_i$ = before(v_1,v_2) :
        - **si** (activité$_j$ = v_1) **alors** successeurs$_j$ ← successeurs$_j$ ∪ v_2
        - **si** (activité$_j$ = v_2) **alors** prédécesseurs$_j$ ← prédécesseurs$_j$ ∪ v_1
      - prédicat$_i$ = after(v_1,v_2) :
        - **si** (activité$_j$ = v_1) **alors** prédécesseurs$_j$ ← prédécesseurs$_j$ ∪ v_2
        - **si** (activité$_j$ = v_2) **alors** successeurs$_j$ ← successeurs$_j$ ∪ v_1
      - prédicat$_i$ = simultaneous(v_1,v_2) :
        - **si** (activité$_j$ = v_1) **alors**
          - prédécesseurs$_j$ ← prédécesseurs$_j$ ∪ v_2
          - successeurs$_j$ ← successeurs$_j$ ∪ v_2
        - **si** (activité$_j$ = v_2) **alors**
          - successeurs$_j$ ← successeurs$_j$ ∪ v_1
          - prédécesseurs$_j$ ← prédécesseurs$_j$ ∪ v_1
      - **autres** : **retourner** exception
    - set(tuple(activité, prédécesseurs, successeurs)) ← set(tuple(activité, prédécesseurs, successeurs)) ∪ tuple(activité$_j$, prédécesseurs$_j$, successeurs$_j$)
- **retourner** set(tuple(activité, prédécesseurs, successeurs))

ALG. 1: Algorithme d'enchaînement

L'algorithme ALG. 1 de transformation des contraintes d'ordonnancement, permettant d'enchaîner les activités du plan de coordination, se déroule de manière suivante. Tout d'abord, l'ensemble des activités à exécuter du plan (set(activité)) est identifié en fonction des paramètres sujets et objets des prédicats d'ordonnancement. Ensuite, pour chaque activité à exécuter, ses prédécesseurs et ses successeurs sont déterminés en fonction de l'expression sémantique des prédicats d'ordonnancement. Considérons par exemple une activité A1. Si A1 est définie comme le paramètre sujet du prédicat d'ordonnancement `before`(), alors l'activité A2 qui est définie comme le paramètre objet de ce prédicat est un des successeurs de A1. L'activité A2 est donc ajoutée à l'ensemble des successeurs de l'activité A1. De manière similaire, si A1 est définie comme le paramètre objet du prédicat `before`(), alors l'activité A2 qui est définie comme le

paramètre sujet de ce prédicat est un des prédécesseurs de A1. A2 est ajoutée à l'ensemble des prédécesseurs de l'activité considérée.

4.2.1c Patrons d'enchaînement

Les propriétés précisant la relation entre une activité et ses prédécesseurs et/ou ses successeurs peuvent être générées lors de la transformation. Cela est fait à partir des contraintes de déclenchement et d'inter-dépendance de données (spécifiées dans le plan).

La transformation détaillée de la relation qui lie une activité à ses prédécesseurs et/ou ses successeurs est basée sur la construction de patrons : séquence, sélection ($\mathsf{ORsplit}, \mathsf{ORjoint}$), itération, synchronisation ($\mathsf{ANDsplit}, \mathsf{ANDjoint}$) et libre ($\mathsf{unordering}$).

Cela permet de vérifier a posteriori la validité de l'exécution du plan.

- Le patron $\mathsf{sequence}(\mathsf{A}_1, \mathsf{A}_2)$ exprime l'enchaînement séquentiel d'une activité spécifique A_1 avec son successeur A_2 ;
- Le patron $\mathsf{ORsplit}(\mathsf{A}_1, \mathsf{set}(\mathsf{A}_2, .., \mathsf{A}_n))$ ou $\mathsf{sequence}(\mathsf{A}_1, \mathsf{A}_2) \vee .. \vee \mathsf{sequence}(\mathsf{A}_1, \mathsf{A}_n)$, exprime l'enchaînement séquentiel d'une activité spécifique A_1 avec l'un de ses successeurs $\mathsf{A}_2, .., \mathsf{A}_n$.
- Le patron $\mathsf{ORjoint}(\mathsf{set}(\mathsf{A}_1, .., \mathsf{A}_{n-1}), \mathsf{A}_n)$ ou $\mathsf{sequence}(\mathsf{A}_1, \mathsf{A}_n) \vee .. \vee \mathsf{sequence}(\mathsf{A}_{n-1}, \mathsf{A}_n)$, exprime l'enchaînement séquentiel de l'un des prédécesseurs $\mathsf{A}_1, .., \mathsf{A}_{n-1}$ d'une activité spécifique A_n avec l'activité A_n.
- Le patron $\mathsf{ANDsplit}(\mathsf{A}_1, \mathsf{set}(\mathsf{A}_2, .., \mathsf{A}_n))$ ou $\mathsf{sequence}(\mathsf{A}_1, \mathsf{A}_2) \wedge .. \wedge \mathsf{sequence}(\mathsf{A}_1, \mathsf{A}_n)$, exprime les enchaînements séquentiels d'une activité spécifique A_1 avec tous ses successeurs $\{\mathsf{A}_2, .., \mathsf{A}_n\}$. L'enchaînement des activités successives est séquentiel ou parallèle.
- Le patron $\mathsf{ANDjoint}(\mathsf{set}(\mathsf{A}_1, .., \mathsf{A}_{n-1}), \mathsf{A}_n)$ ou $\mathsf{sequence}(\mathsf{A}_1, \mathsf{A}_n) \wedge .. \wedge \mathsf{sequence}(\mathsf{A}_{n-1}, \mathsf{A}_n)$ exprime les enchaînements séquentiels de tous les prédécesseurs $\mathsf{A}_1, .., \mathsf{A}_{n-1}$ d'une activité spécifique A_n avec l'activité A_n.
- Le patron $\mathsf{iteration}(\mathsf{A}_1, .., \mathsf{A}_n)$ exprime la répétition d'un enchaînement séquentiel des activités $\mathsf{A}_1, .., \mathsf{A}_n$.
- Le patron $\mathsf{unordering}(\mathsf{A}_1, .., \mathsf{A}_n)$ exprime l'enchaînement dynamique des activités $\mathsf{A}_1, .., \mathsf{A}_n$ (c.-à-d. qu'il n'y a pas de contraintes spécifiées au préalable). Les activités ne sont donc pas ordonnées ; elles s'exécuteront parallèlement ou dans n'importe quel ordre.

4.2.2 Regroupement des contraintes selon différents critères

Les contraintes spécialisées sont associées aux activités en tant que pré ou post-condition de l'appel. Les contraintes locales et globales sont associées aux activités en tant que propriétés d'évaluation. Étant donnée une spécification d'une activité, il est donc nécessaire de regrouper toutes les contraintes associées selon certains critères ; cela permet d'effectuer les algorithmes d'évaluation appropriés. Pour cela, nous nous intéressons :

- aux contraintes à évaluer conjointement : le regroupement des contraintes, utilisant le temps d'évaluation, sert à l'évaluation des contraintes locales d'une activité.
- aux contraintes à évaluer séparément : le regroupement des contraintes, utilisant le temps d'évaluation, sert à l'évaluation des contraintes globales du plan.

Un critère **C** correspond à différents types de propriété **P1**, **P2**, .., **Pn**. La classification de contraintes selon **C** vise à les "dispatcher" en différents groupes ; chaque groupe correspond soit à un type de propriétés **P1**, **P2**, .., **Pn**, soit à une combinaison de ces types. Les contraintes qui ne respectent pas le critère **C** sont ajoutées à un groupe par défaut marqué $\natural$.

Cette section présente d'abord les critères de regroupement, puis elle décrit comment regrouper les contraintes selon un critère.

4.2.2a Critères de regroupement

evalSkill : Ce critère correspond aux propriétés de compétence de contrainte qui sont caractérisées de la manière suivante :

$$\mathsf{Property} :\!- \mathsf{tuple}(\mathit{propertyName} : \mathit{SkillInfo}, \mathit{propertyValue} : \mathsf{Formula}) \tag{4.2.3}$$

où $\mathsf{SkillInfo} = \{\mathsf{ordering}, \mathsf{firing}, \mathsf{data}, \mathsf{safety}, \mathsf{authentication}, \mathsf{autorization}, \mathsf{integrity}, \mathsf{nonrepudiation}, \mathsf{adaptation}, \mathsf{rendering}, \mathsf{control}^\natural\}$ représente les noms d'aspects pour lesquels les prédicats constituant une contrainte sont définis (ordering, firing, etc.). Les propriétés de cette forme sont utiles pour vérifier la cohérence des contraintes d'un même aspect. Ainsi, elles sont utiles pour reconnaître des nouveaux prédicats d'un aspect.

Exemple. Property :– tuple(*propertyName* : ordering, *propertyValue* : `simultaneous`()) : Il s'agit d'une propriété dont le nom (ordering) appartient au domaine SkillInfo et dont la valeur (`simultaneous`()) appartient au domaine Formula. cela signifie que `simultaneous`() est un prédicat prédéfini pour l'aspect ordonnancement. Le nom de la propriété (ordering) permet d'instancier un évaluateur spécialisé à l'évaluation d'une contrainte d'ordonnancement lors de l'exécution.
De même, Property :– tuple(*propertyName* : ordering, *propertyValue* : *Timing*()) signifie que le prédicat Timing() représente la propriété Timing prédéfinie pour cet aspect. Le nom de propriété (ordering) permet d'instancier les paramètres des contraintes d'ordonnancement lors de son évaluation.

evalPosition : Ce critère correspond aux propriétés de positionnement de l'évaluation de contrainte qui sont caractérisées par :

$$\begin{aligned}
&\mathsf{Property} :\!- \mathsf{tuple}(\mathit{propertyName} : \mathsf{LocationInfo}, \mathit{propertyValue} : \mathsf{Formula})\\
&\mathsf{Property} :\!- \mathsf{tuple}(\mathit{propertyName} : \mathsf{TimeInfo}, \mathit{propertyValue} : \mathsf{Formula})\\
&\mathsf{Property} :\!- \mathsf{tuple}(\mathit{propertyName} : \mathsf{EventInfo}, \mathit{propertyValue} : \mathsf{Formula})
\end{aligned} \tag{4.2.4}$$

où :

- Le domaine LocationInfo définit les valeurs représentant l'identificateur d'activités auxquelles s'attache la contrainte à satisfaire ($\subset \mathsf{dom}(\mathsf{activityId})$). Par défaut, des paramètres sujets de la portée de chaque contrainte sont décrits dans la spécification de telles activités.
- Le domaine $\mathsf{TimeInfo} = \{\mathsf{pre}, \mathsf{post}, \mathsf{during}^\natural\}$ définit les valeurs représentant le moment où l'évaluation d'une contrainte se produit par rapport à un événement déclenchant spécifique.
- Le domaine $\mathsf{EventInfo} = \{\mathsf{activation}, \mathsf{execution}^\natural, \mathsf{evaluation}\}$ se compose des valeurs représentant des événements clefs qui déclenchent l'évaluation d'une contrainte : l'activation d'une preuve d'exécution, l'exécution d'appel ou l'évaluation d'une autre contrainte. Les valeurs de ce domaine sont utilisées comme des noms des propriétés de positionnement de contraintes.

Une contrainte vérifie le critère de regroupement evalPosition si et seulement si elle est étiquetée par une propriété de chacun de types ci-dessus. Ces propriétés sont utilisées pour vérifier si cette contrainte, associée en tant que pré-condition ou post-condition de l'appel d'une activité

spécifique, respecte les règles d'association aux activités (voir la section 3.4.1).

Exemple. *A1* : before(A1, A2), *pre* : before(A1, A2), et *execution* : before(A1, A2) sont trois instances de propriété *evalPosition*. Ce sont des propriétés définies dans la spécification de l'activité A1. Nous disons que le prédicat before(A1, A2) est étiqueté à la fois par trois valeurs A1 (appartenant au domaine LocationInfo), pre (appartenant au domaine TimeInfo) et execution (appartenant au domaine EventInfo). Ces valeurs signifient que le prédicat before(A_1, A_2) est à évaluer dans le cadre de l'activité A1, avant l'exécution de l'appel correspondant.

evalScope : Ce critère correspond aux propriétés de l'expression de l'évaluation de contrainte qui sont caractérisées par :

$$\text{Property} :- \text{tuple}(\textit{propertyName} : \textit{ScopeInfo}, \textit{propertyValue} : \text{Formula}) \tag{4.2.5}$$

où le domaine ScopeInfo = {static♮, dynamic} se compose de valeurs précisant l'instanciation des paramètres d'une contrainte. La valeur statique signifie que les paramètres de la contrainte sont instanciés avant son évaluation (évaluation sur la portée statique). La valeur (dynamic) signifie que les paramètres de la contrainte sont instanciés au moment de son évaluation (évaluation sur la portée dynamique).

Les propriétés de cette forme sont utiles pour instancier correctement les paramètres d'une contrainte, ainsi que pour rendre prioritaire l'évaluation d'une contrainte.

4.2.2b Algorithme de regroupement

L'algorithme ALG. 2 décrit ci-dessous permet de regrouper des contraintes selon les critères considérés (domaines de propriétés) : compétence de l'évaluation, position de l'évaluation et expression de l'évaluation. Pour chaque contrainte, il faut tout d'abord lire la valeur de la propriété représentant le critère de classification. Ensuite, en fonction de cette valeur, il faut : soit ajouter la contrainte à un groupe de contraintes de même critère existant, soit créer un nouveau groupe de contraintes et y ajouter la contrainte.

*[Input : **set(formule)**, un ensemble de contraintes à évaluer ; cette valeur est de type set(Formula)*
***classification**, une de trois valeurs evalSkill, evalPosition, evalScope représentant un critère de regroupement]*
*[Output : **list(critère : $\text{set}_{\text{critère}}$(formule))**, une liste de contraintes qui sont classifiées selon un critère donné]*
classify : (*constraintset* : set(Formula)) × (*groupcriteria* : String) → (choice(list(*evalSkill* : set(Formula)), list(*evalPosition* : set(Formula)), list(*evalScope* : set(Formula))))

pour chaque contrainte ∈ set(formule) **faire**
  critère $\overset{classification}{\leftarrow}$ getInfo(contrainte) *[Lire sa propriété de critère]*
  si (∃ $\text{set}_{\text{critère}}$(formule))
    alors *[Ajoute la contrainte à un groupe de contraintes de même critère déjà crée]*
      $\text{set}_{\text{critère}}$(formule) ← $\text{set}_{\text{critère}}$(formule) ∪ (formule = contrainte)
    sinon *[Crée un groupe de contraintes de critère correspondant et y ajoute la contrainte]*
      $\text{set}_{\text{critère}}$(formule) ← ϕ
      $\text{set}_{\text{critère}}$(formule) ← $\text{set}_{\text{critère}}$(formule) ∪ (formule = contrainte)
retourner list(critère : $\text{set}_{\text{critère}}$(formule))

ALG. 2: Processus de regroupement des contraintes

4.2.3 Analyse de contraintes

L'analyse syntaxique, qui suit les règles de définition des formules bien formées (voir la section 3.2.3c), fournit des propriétés utiles pour leur évaluation. Ces propriétés sont caractérisées selon deux critères suivants :

evalStructure : Ce critère correspond aux propriétés de structure de contrainte qui sont caractérisées de la manière suivante :

$$\mathsf{Property} :- \mathsf{tuple}(\mathit{propertyName} : \mathit{StructureInfo}, \mathit{propertyValue} : \mathsf{Formula}) \tag{4.2.6}$$

où le domaine $\mathsf{StructureInfo} = \{\mathsf{atomic}^\natural, \mathsf{composite}, \mathsf{nested}\}$ caractérise les structures d'une formule bien formée (à base de prédicats) définissant une contrainte. La valeur atomic signifie que la formule est définie par un prédicat. La valeur composite signifie que la formule est définie de manière non récursive par des prédicats liés par des connecteurs logiques. La valeur nested signifie que la formule est définie de manière récursive par des prédicats liés par des connecteurs logiques.

Les propriétés de ce type sont utilisées pour choisir les évaluateurs appropriés. Elles permettent également de faciliter l'évaluation partielle d'une contrainte globale, surtout dans le cas où la portée de cette contrainte est dynamiquement reconstruite après l'évaluation de chaque contrainte spécialisée constituante.

evalSatisfaction : Ce critère correspond aux propriétés de niveau de satisfaction de contrainte qui sont caractérisées par :

$$\mathsf{Property} :- \mathsf{tuple}(\mathit{propertyName} : \mathit{SatisfactionInfo}, \mathit{propertyValue} : \mathsf{Formula}) \tag{4.2.7}$$

où le domaine $\mathsf{SatisfactionInfo} = \{\mathsf{partial}, \mathsf{total}^\natural\} \subset \mathsf{dom}(\mathsf{String})$ caractérise les besoins de satisfaction d'une contrainte spécialisée d'une activité ; ceux-ci sont considérés au niveau d'une contrainte globale du plan ou au niveau d'une contrainte locale de cette activité. La valeur total signifie que la non-satisfaction de cette contrainte implique la non-satisfaction de la contrainte globale ou locale dont elle fait partie.

Les propriétés de ce type sont utiles pour l'adaptation de l'évaluation d'une contrainte globale où les contraintes spécialisées constituantes ne sont pas évaluées conjointement. Elles sont aussi utiles pour optimiser le traitement d'exception lors de l'évaluation d'une contrainte locale.

4.2.4 Priorités d'évaluation

Les contraintes d'un plan de coordination sécurisée d'activités concernent différents aspects : coordination, sécurité, adaptation, etc. L'évaluation de ces contraintes respecte leur priorité. Cette section présente le critère de priorité de l'évaluation de contrainte et indique comment rendre une contrainte prioritaire.

evalPriority Ce critère correspond aux propriétés de priorité de l'évaluation de contrainte qui sont caractérisées de la manière suivante :

$$\mathsf{Property} :- \mathsf{tuple}(\mathit{propertyName} : \mathit{PriorityInfo}, \mathit{propertyValue} : \mathsf{Formula}) \tag{4.2.8}$$

où le domaine PriorityInfo ⊂ dom(Integer) énumère les priorités de l'évaluation d'une contrainte. Dans ce domaine, la valeur la plus grande signifie la plus haute priorité considérée. La valeur 0 signifie la plus basse priorité considérée.

Les propriétés de ce type sont utiles pour l'évaluation d'une collection de contraintes de différentes compétences (critère `evalSkill`) ou l'évaluation d'une collection de contraintes de différentes expressions d'évaluation(critère `evalScope`).

Définition de la priorité d'évaluation en se basant sur la compétence des contraintes : Par défaut, les contraintes de différents aspects possèdent des priorités qui sont établies par l'algorithme ALG. 3. Cet algorithme utilise le critère `evalSkill`. Les contraintes des aspects contrôle possèdent les plus hautes priorités. Les contraintes des aspects sécurité possèdent les priorités moyennes. Les contraintes des aspects coordination possèdent les plus basses priorités.

*[Input : **set(compétence)**, ensemble des compétences à mettre en priorité]*
*[Output : **set(tuple(priorité, compétence))** ensemble des compétences avec leur priorité correspondante]*
skillprioritysetting : (*skills : set(SkillInfo)*) → (set(tuple(PriorityInfo,SkillInfo)))
- **pour chaque** compétence ∈ set(compétence) **faire**
  - **selon que**
    - compétence = adaptation : priorité ← 8
    - compétence = safety : priorité ← 7
    - compétence = authentication : priorité ← 6
    - compétence = integrity : priorité ← 5
    - compétence = autorisation : priorité ← 4
    - compétence = nonrepudiation : priorité ← 3
    - compétence = datainterdependency : priorité ← 2
    - compétence = ordering : priorité ← 1
    - compétence = firing : priorité ← 0
- **retourner** set(tuple(priorité, compétence))

ALG. 3: Définition des priorités d'évaluation par défaut pour les aspects coordination et sécurité

*[Input : **set(expression)**, ensemble des expression de l'évaluation à mettre en priorité]*
*[Output : **set(tuple(priorité, expression))** ensemble des expressions de l'évaluation avec leur priorité correspondante]*
expressionprioritysetting : (*expressions : set(ScopeInfo)*) → (set(tuple(PriorityInfo,ScopeInfo)))
- **pour chaque** compétence ∈ set(compétence) **faire**
  - **selon que**
    - expression = dynamic : priorité ← 1
    - expression = static : priorité ← 0
- **retourner** set(tuple(priorité, expression))

ALG. 4: Définition des priorités d'évaluation par défaut pour les différentes expressions de l'évaluation de contrainte

Définition de la priorité en se basant sur l'expression de l'évaluation de contrainte : Par défaut, la priorité pour les contraintes de même aspect et de même l'expression d'évaluation est établie par l'algorithme ALG. 4. Cet algorithme utilise le critère `evalScope`. Les contraintes devant être évaluées sur les portées dynamiques sont rendues prioritaires ; cela permet de fixer

a priori les portées d'évaluation pour les autres contraintes devant être évaluées sur les portées statiques.

4.3 Algorithme d'évaluation d'une contrainte

Une contrainte est représentée par une formule logique bien formée. Cette dernière est définie à partir de termes (constante, variable, fonction) et de prédicats.

Les prédicats et les fonctions (avec arité) définissant la formule sont considérés comme des *opérateurs de la contrainte*. Les opérateurs de la contrainte définissent alors des valeurs satisfaisant la contrainte.

Les domaines représentant les valeurs potentielles des paramètres (d'entrée, de sortie) de tous les opérateurs de la contrainte constituent la *portée de la contrainte*[1]. La portée de la contrainte définit des valeurs possibles pour l'évaluation de la contrainte ; elle représente donc les paramètres de la contrainte.

Une contrainte est dite satisfaite si une *affectation* de valeurs à ses paramètres est trouvée dans sa portée conformément à ses opérateurs. L'affectation respecte des règles suivantes :

- un symbole de constante[2] prend pour valeur la constante elle-même ;
- un symbole de variable[3] prend une valeur typée ;
- un symbole de fonction (ou opération) n-aire prend une valeur typée ;
- un symbole de prédicat n-aire (ou connecteur, quantificateur) prend une valeur booléenne.

Dans notre contexte, une telle évaluation est interprétée selon les deux cas suivants :

1. **Évaluation sur la portée statique** : cette évaluation permet d'évaluer une contrainte dont les paramètres sont instanciés **a priori** : soit au moment de spécifier cette contrainte, soit lors de l'évaluation sur la portée dynamique d'une autre contrainte. Le résultat de cette évaluation indique si l'instanciation des paramètres de la contrainte est satisfaite ou non. Par exemple, l'évaluation des contraintes d'ordonnancement ou d'authentification.
2. **Évaluation sur la portée dynamique** : cette évaluation permet d'évaluer une contrainte dont :
   - la valeur booléenne souhaitée du résultat d'évaluation a été déclarée a priori ;
   - les paramètres doivent être instanciés ou mis à jour **au cours de l'évaluation**.

   Le résultat de cette évaluation indique si l'instanciation ou la mise à jour des paramètres de la contrainte est satisfaite ou non, par rapport au résultat souhaité. Exemples : l'évaluation d'appel ou l'évaluation de contraintes d'inter-dépendance de données.

Il est important de rappeler qu'une contrainte C doit être évaluée dans le cadre de l'exécution d'une activité A du plan. Du point de vue sémantique, un paramètre de C prend une valeur conformément à la valeur que les autres paramètres de C ont pris. Du point de vue pratique, les valeurs potentielles des paramètres de C peuvent être extraites de la spécification de l'activité A ou d'une autre activité. Elles peuvent également être produites dans le cadre de l'exécution de l'activité A ou dans le cadre de l'exécution d'une autre activité.

Afin de faciliter l'instanciation correcte des paramètres, la portée de la contrainte est (re)construite au moment de l'évaluation ; cela est fait en regroupant les paramètres des opérateurs de

[1] Chaque opérateur est défini sur un ou plusieurs domaines de cette portée.
[2] Il est également considéré comme un symbole de fonction 0-aire.
[3] Il est également considéré comme un symbole de prédicat 0-aire.

la contrainte en deux groupes : les paramètres sujets (dit *sujets de contrainte*), et les paramètres objets (dit *objets de contrainte*)[4]. Ce regroupement permet de préciser, du point de vue sémantique et pratique, la manière d'instancier des paramètres.

Étant donné un opérateur de la contrainte, l'affectation des valeurs à ses paramètres sujets est conforme à l'affectation des valeurs à ses paramètres objets. La contrainte est évaluée à *VRAI* si l'affectation des valeurs satisfaites aux paramètres de tous les opérateurs de la contrainte est trouvée dans cette portée. Sinon, la contrainte est évaluée à *FAUX* (voir la figure 4.7).

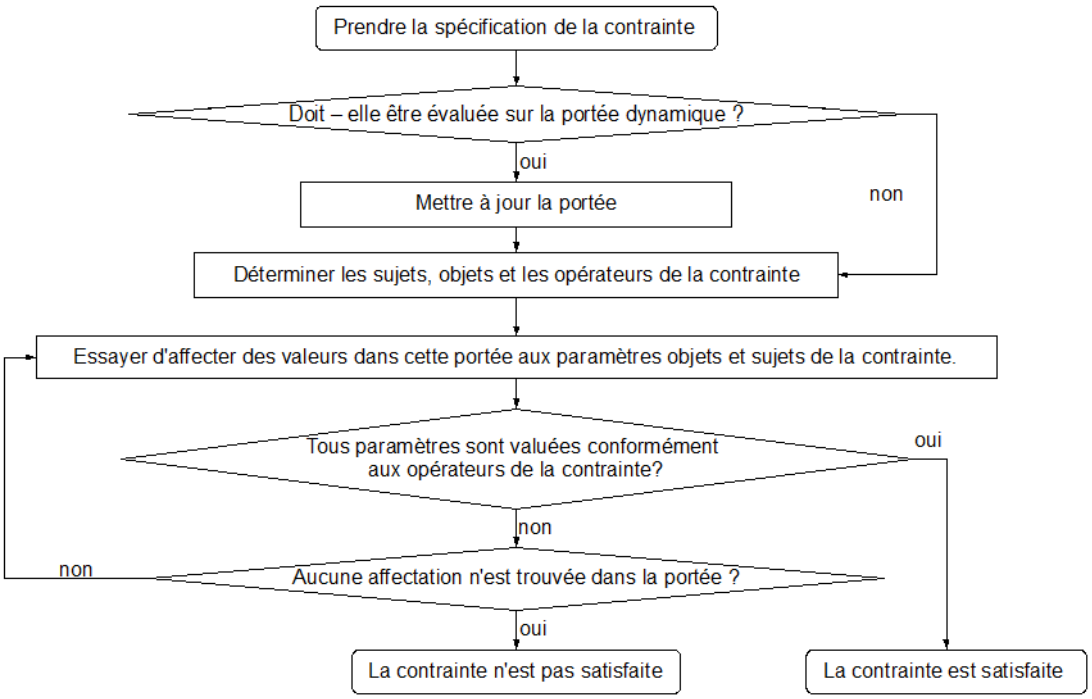


FIG. 4.7 – Logigramme de l'évaluation d'une contrainte

Notre algorithme d'évaluation est brièvement décrit comme suit. Initialement, les paramètres sujets et objets des opérateurs de la contrainte ne sont pas encore valués[5] (l'affectation est vide). Successivement, l'affectation de(s) valeur(s) au(x) paramètre(s) sujet(s) est effectuée en accord avec l'affectation de(s) valeur(s) au(x) paramètre(s) objet(s) correspondant(s). Un paramètre (pas encore valué) ne prend une valeur que si cette valeur est conforme aux paramètres déjà valués. Si toutes les valeurs potentielles d'un paramètre sujet (ou objet) ne sont pas conformes aux valeurs que les autres paramètres ont déjà pris (c.-à-d. ce paramètre est non affectable), ces étapes sont répétées pour toutes les valeurs potentielles restantes du paramètre précédemment valué. En fin, si tout paramètre est valué, l'affectation des valeurs satisfaites est complète et les contraintes sont évaluées à *VRAI* (à *FAUX* sinon).

Cet algorithme est implanté de manière récursive par des fonctions evaluate, assign, verify qui sont représentées dans la suite de cette section.

4.3.1 Fonction "evaluate"

Cette fonction a pour but d'évaluer une contrainte qui est définie par une formule bien formée (voir l'algorithme 5). Elle utilise des informations spécifiant la contrainte ; ces informations

[4] Dans notre contexte, les prédicats et les fonctions prédéfinies sont unaires ou binaires. D'une manière générale, les prédicats et les fonctions n-aires sont transformables en fonctions unaires ou binaires.

[5] Ces sujets et objets de la contrainte sont donc représentés par la valeur "NULL" (?) qui signifie une information inconnue à l'instance.

sont extraites de la spécification de l'activité à laquelle la contrainte est associée et/ou des preuves d'exécution des activités concernées. Dans le cas d'évaluation sur la portée dynamique, elle reconstruit la portée de la contrainte. Ensuite, à partir de la portée de la contrainte, elle détermine les domaines de valeurs potentielles des paramètres sujets et objets de contrainte (en appelant les fonctions getConstraintSubjects et getConstraintObjects). Elle détermine également les opérateurs de la contrainte (en appelant la fonction getOperators). En fin, elle appelle la fonction assign (voir la section 4.3.2) afin de chercher dans la portée une affectation des valeurs à tous les paramètres de contrainte. En fonction du résultat renvoyé par la fonction assign, la fonction evaluate rend le résultat de l'évaluation de contrainte. Si tout paramètre est valué conformément à tous opérateurs de la contrainte (c.-à-d. une affectation complète est trouvée), la contrainte est évaluée à *VRAI* . Sinon, la contrainte est évaluée à *FAUX* .

*[Input : **F**, la formule représentant une contrainte]*
*[Output : retourne **VRAI** si une solution a été trouvée ; retourne **FAUX** sinon]*
evaluate : (*constraint* : Formula) → (*evaluatedResult* : Boolean)
  si (evalScope = dynamic) **alors** updateScope(F)
  set(cSubj) ← getConstraintSubjects(F)
  set(cObj) ← getConstraintObjects(F)
  set(cOp) ← getConstraintOperators(F)
  solution ← assign(set(cSubj), set(cObj), set(cOp), ϕ, ϕ)
  si (solution ≠ exception)
    alors retourner VRAI
    sinon retourner FAUX

Alg. 5: Fonction "evaluate"

Exemple. Soit la contrainte d'ordonnancement
`before(Activity(activityId(A3)), Activity(activityId(A4)))`
Le paramètre sujet est le temps de fin d'exécution de l'activité A3. Le domaine de valeurs potentielles à attribuer à ce paramètre est un sous domaine du domaine *Timing* (voir la section 3.3.1). Il contient trois valeurs : une valeur extraite à partir de la spécification de A3, une valeur extraite à partir du journal de coordination de A3 et une valeur observée au cours de l'exécution d'A3.
Le paramètre objet est le temps de début d'exécution de l'activité A4. Le domaine de valeurs potentielles à attribuer à ce paramètre est aussi un sous domaine du domaine *Timing*. Il contient deux valeurs : une valeur extraite à partir de la spécification de A4 et une valeur extraite à partir du journal de coordination de A4.
L'opérateur de la contrainte est une opération permettant de comparer si la valeur représentant le temps de fin d'exécution d'une activité est plus petite que celle représentant le temps de début d'exécution d'une autre activité.

4.3.2 Fonction "assign"

Cette fonction a pour but de chercher (dans la portée de la contrainte) une affectation des valeurs à tous les paramètres de la contrainte conformément à tous opérateurs de la contrainte. Elle est décrite par l'algorithme Alg. 6.

Tout d'abord, la fonction détermine des paramètres qui sont présents à la fois en tant que paramètres sujet et paramètres objet (cCommonScope).

*[Input : **set(cSubj)** ensemble de sujets de la contrainte avec leurs domaines de valeurs construites pré-déterminés,*
***set(cObj)** ensemble d'objets de la contrainte avec leurs domaines de valeurs construites pré-déterminés,*
***set(cOp)** ensemble des opérateurs de la contrainte,*
***csAssigment** affectation initiale pour les sujets de la contrainte, et*
***coAssigment** affectation initiale pour les objets de la contrainte]*
[Output : retourne une solution si une affectation complète a été trouvée ; retourne une exception sinon]

assign : (*constraintSubjects* : set(T)) × (*constraintObjects* : set(T))× (*constraintOperators* : set(Operation)) × (*subjectAssignment* : set(T)) × (*objectAssignment* : set(T)) → (*solution* : set(T))

- **si** (set(cSubj) = ϕ **OU** set(cObj) = ϕ) **alors retourner** tuple(csAssignment, coAssignment)
- cCommonScope ← set(cSubj) ∩ set(cObj)
- consSubj ← { cSubj ∈ set(cSubj) | ∀ cs ∈ set(cSubj) ‖ dom(cSubj) ‖ ≤ ‖ dom(cs) ‖ }
- **si** (consSubj ∈ cCommonScope)
  - **alors**
    - consObj ← consSubj
    - **pour chaque** $value_j$ ∈ dom(consObj) **faire**
      - csAssignment ← csAssignment ∪ $value_j$
      - coAssignment ← coAssignment ∪ $value_j$
      - **si** (verify(csAssignment, coAssignment, set(CO))) **alors**
        - set(cSubj) ← set(cSubj) \ (cSubj = consSubj)
        - set(cObj) ← set(cObj) \ (cObj = consObj)
        - solution ← assign(set(cSubj), set(cObj), set(cOp), csAssignment, coAssignment)
        - **si** (solution ≠ exception) **alors retourner** solution
      - csAssignment ← csAssignment \ $value_j$
      - coAssignment ← coAssignment \ $value_j$
  - **sinon**
    - consObj ← { cObj ∈ set(cObj) | ∀ co ∈ set(cObj) ‖ dom(cObj) ‖ ≤ ‖ dom(co) ‖ }
    - **pour chaque** $value_i$ ∈ dom(consSubj) **faire**
      - csAssignment ← csAssignment ∪ $value_i$
      - **pour chaque** $value_j$ ∈ dom(consObj) **faire**
        - coAssignment ← coAssignment ∪ $value_j$
        - **si** (verify(csAssignment, coAssignment, set(CO))) **alors**
          - set(cSubj) ← set(cSubj) \ (cSubj = consSubj)
          - set(cObj) ← set(cObj) \ (cObj = consObj)
          - solution ← assign(set(cSubj), set(cObj), set(cOp), csAssignment, coAssignment)
          - **si** (solution ≠ exception) **alors retourner** solution
        - coAssignment ← coAssignment \ $value_j$
      - csAssignment ← csAssignment \ $value_i$
- **retourner** exception

ALG. 6: Fonction "assign"

Ensuite, un paramètre sujet qui n'a pas pris de valeur est choisi (consSubj) ; il s'agit du paramètre dont la cardinalité de valeurs potentielles est la plus petite.

Si ce paramètre est présent également en tant que paramètre objet, c'est lui qui est choisi comme le paramètre objet à retenir (consObj). Ensuite, la fonction assign essaie d'affecter une valeur (dans le domaine de valeurs potentielles à affecter) à ce paramètre, en tant que paramètre

sujet et paramètre objet. Elle appelle la fonction verify (voir la section 4.3.3) pour savoir si cette valeur est conforme aux opérateurs de la contrainte, ainsi qu'aux paramètres précédemment valués. Si c'est le cas, ce paramètre est compté à la fois comme le paramètre sujet valué et le paramètre objet valué. La fonction assign est récursive, de manière à effectuer une autre affectation d'un autre paramètre non valué. Le cas échéant, la fonction réessaie avec une autre valeur à ce paramètre (cette valeur appartient au domaine de valeurs potentielles de ce paramètre). S'il est impossible d'affecter une valeur à ce paramètre, la fonction assign renvoie une exception.

Si ce paramètre n'est pas présent en tant que paramètre objet, un paramètre objet non valué, dont la cardinalité est la plus petite parmi les autres, est choisi. Ensuite, la fonction assign essaie d'affecter une valeur au paramètre sujet (cette valeur appartient au domaine de valeurs potentielles de ce paramètre). De même, cette fonction affecte une valeur au paramètre objet choisi. Elle appelle la fonction verify pour savoir si cette valeur est conforme aux opérateurs de la contrainte, ainsi qu'aux paramètres précédemment valués. Cette fonction continue son exécution comme dans le cas précédent.

4.3.3 Fonction "verify"

Cette fonction a pour but faire exécuter les opérateurs de la contrainte. En fonction des résultats renvoyés, elle va déterminer la conformité d'une affectation des valeurs par rapport à ces opérateurs. Elle est décrite par l'algorithme Alg. 7.

*[Input : **csAssigment** une affectation initiale pour les sujets de contrainte, **coAssigment** une affectation initiale pour les objets de contrainte et **set(cOp)** un ensemble des opérateurs de la contrainte]*

*[Output : retourne **VRAI** si un opérateur de contrainte est exécutable ; retourne **FAUX** sinon]*

verify : (*subjectAssignment* : set(T)) × (*objectAssignment* : set(T)) × (*constraintOperators* : set(Operation))
→ (*solution* : Boolean)

```
pour chaque cOp_i ∈ dom(cOp) faire
    si ((cOp_i'sSubjectParam ⊂ csAssignment) ∧ (cOp_i'sObjectParam ⊂ coAssignment))
    alors
        si (csAssignment et coAssignment rendent cOp_i non exécutable) alors
            retourner FAUX
retourner VRAI
```

Alg. 7: Fonction "verify"

4.3.4 Contraintes évaluables

Une contrainte est dite *évaluable* par cet algorithme si et seulement si :

- les domaines de valeurs potentielles des paramètres sont pré-déterminés par le système de types ;
- les valeurs que prennent les paramètres de la contrainte sont bien typées ; et
- les opérateurs de la contrainte sont prédéfinis sur les domaines de valeurs potentielles des paramètres de la contrainte.

4.3.5 Traitement d'exception

L'évaluation à *FAUX* d'une contrainte est capturée comme une exception. Par défaut, cette dernière annule l'exécution de l'activité courante. Néanmoins, nous laissons la capacité de traiter une telle exception en terme des propriétés d'exception prédéfinies ou des intervention au cours

de l'exécution (voir l'algorithme ALG. 8). Le traitement d'exception permet donc d'adapter la suite de l'exécution du processus d'évaluation : réinitialiser le processus, reprendre le processus à l'état d'exécution avant le moment où l'exception est survenue, ou bien continuer le processus sans prendre en compte l'exception.

Le traitement d'une exception est conforme à la contrainte globale (politique) qui contient la contrainte spécialisée non satisfaite. Il se base sur des preuves d'exécution archivées. Il dépend également d'instructions interactives pendant l'exécution.

HandleException : (*constraint* : Formula) → (*treatmentresult* : Boolean)
selon que
[Compensation : Continuer avec exception]
intervention interne ∨ propriété de contrôle : **retourner VRAI**
[Compensation : Réessayer l'évaluation jusqu'au moment où elle soit VRAI ou timeout]
intervention externe :
répéter
result $\overset{choice(dynamic,static)}{\longleftarrow}$ evaluate(contrainte)
jusqu'à ce que (result = **VRAI** ∨ timeout)
retourner result
[Annulation : Arrêt + notifier l'exception]
par défaut, timeout ou pas d'intervention :
notifier l'exception
retourner FAUX

ALG. 8: Algorithme de traitement d'exception

4.3.6 Processus et transition d'états

$$constraint-evaluation ::= treatment \mid (preparation \wedge treatment \wedge termination)\ ; \tag{4.3.1}$$
$$preparation ::= evaluable-checking \mid (evaluable-checking \wedge exception-treatment) \tag{4.3.2}$$
$$\mid UNDER\_CONSIDERATION \mid WAITING \mid READY\ ;$$
$$treatment ::= evaluation-function-invocation \mid (evaluation-function-invocation \wedge exception-treatment) \tag{4.3.3}$$
$$\mid ACTIVE\ ;$$
$$termination ::= ABORT \mid COMMIT\ ; \tag{4.3.4}$$
$$exception-treatment ::= PAUSE \mid COMPENSATIVE\ ; \tag{4.3.5}$$
$$evaluable-checking ::= VRAI \mid FAUX\ ; \tag{4.3.6}$$

FIG. 4.8 – Règles de contrôle de l'évaluation d'une contrainte

La figure 4.8 illustre les règles de contrôle du processus d'évaluation d'une contrainte par un évaluateur. Ce processus se compose également de trois phases : *preparation*, *treatment* et *termination*. Les phases de préparation et de terminaison sont optionnelles, dans le cas où le processus d'exécution de l'évaluation consiste à appeler un autre évaluateur spécifique (voir la règle 4.3.1). La figure 4.9 illustre l'automate états-transitions d'une contrainte durant son évaluation.

Dans la phase de préparation, les valeurs potentielles des paramètres de la contrainte sont déterminées avant la vérification de leur satisfaction. Cette phase concerne les états d'exécution : UNDER_CONSIDERATION, WAITING, READY (voir la règle 4.3.2). Initialement, l'état d'évaluation

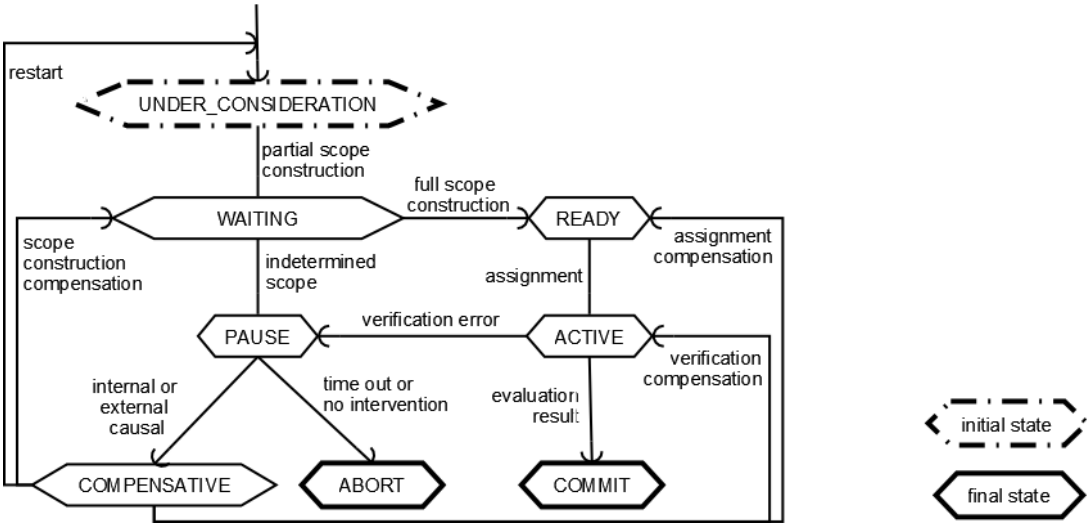


FIG. 4.9 – Automate états-transitions de l'évaluation d'une contrainte

d'une contrainte est UNDER_CONSIDERATION. Cet état devient WAITING durant la recherche des valeurs potentielles de ses paramètres. Une fois que toutes les valeurs potentielles sont déterminées, l'état d'évaluation de la contrainte devient READY.

Dans la phase de traitement, la satisfaction des valeurs affectées est vérifiée. Cette phase concerne les états d'exécution READY, ACTIVE (voir les règles 4.3.2 et 4.3.3). Une fois que l'affectation des valeurs aux paramètres de la contrainte est réalisée, l'état d'évaluation est passé à ACTIVE. Dans cet état, l'évaluation de la contrainte sur les paramètres valués est effectuée.

Dans la phase de terminaison, la contrainte est évaluée en se basant sur le résultat de vérification de la satisfaction des valeurs affectées. Cette phase concerne les états d'exécution ACTIVE, COMMIT, ABORT (voir les règles 4.3.3 et 4.3.4). Une fois que la fonction d'évaluation est terminée, le résultat de l'évaluation est renvoyé et l'état de l'évaluation est l'état final COMMIT.

Le processus d'exécution qui se termine dans l'état COMMIT rend le résultat de l'évaluation correspondant au résultat de vérification de la satisfaction des valeurs affectées. Le processus d'exécution qui se termine dans l'état ABORT signifie l'évaluation à *FAUX* de la contrainte.

L'impossibilité (i) de la construction des domaines de valeurs potentielles à affecter aux paramètres de la contrainte, (ii) de l'affectation des valeurs aux paramètres de la contrainte, ou (iii) de vérifier la satisfaction des valeurs affectées déclenche le traitement d'exception. Il est possible de compenser partiellement ou totalement le processus d'exécution de l'évaluation, si une exception apparaît durant l'évaluation. L'état d'exécution de l'évaluation est passé à PAUSE pour attendre une intervention de compensation. En fonction de l'intervention, cet état est passé à COMPENSATIVE, puis il peut être replacé à UNDER_CONSIDERATION (pour la compensation totale) ou un autre état du processus (pour la compensation partielle). Dans l'impossibilité de construire les domaines de valeurs potentielles des paramètres de la contrainte, l'état d'exécution du processus de l'évaluation passe de PAUSE à WAITING ou à UNDER_CONSIDERATION. Si l'affectation des valeurs aux paramètres de la contrainte est impossible, l'état d'exécution du processus de l'évaluation passe de PAUSE à READY ou à UNDER_CONSIDERATION. Si la satisfaction des valeurs affectées n'est pas vérifiée, l'état d'exécution du processus de l'évaluation passe de PAUSE à READY ou à UNDER_CONSIDERATION. Le cas échéant (pas d'intervention ou dépassement du délai d'attente), le processus d'évaluation se termine dans l'état ABORT. Cet état signifie l'évaluation à *FAUX* de la contrainte.

4.4 Évaluation d'une contrainte spécialisée

Une contrainte spécialisée est définie à partir des prédicats prédéfinis d'un aspect considéré. Les opérateurs d'une contrainte spécialisée sont des prédicats dont les conditions de satisfaction ont été définies dans le chapitre modèle. Une contrainte spécialisée est évaluable selon notre algorithme.

Pour l'exécution d'un plan de coordination sécurisée d'activités, nous nous intéressons à l'évaluation des contraintes spécialisées qui sont regroupées selon le critère *evalSkill* (voir la section 4.2.2). Ces contraintes servent au même aspect du plan de coordination sécurisée. Elles doivent être conjointement satisfaites soit avant, soit après l'exécution d'un appel dans le cadre d'une activité spécifique. L'évaluation de telles contraintes consiste en l'évaluation de la *conjonction* de résultats de chacune.

L'algorithme ALG. 9 illustre le processus d'évaluation d'un ensemble des contraintes spécialisées ; ces dernières sont étiquetées par la même propriété de compétence dont le nom appartient au domaine SkillInfo.

*[Input : **set(formule)**, un ensemble des contraintes de même aspect devant être évaluées]*
[Output : résultat final de l'évaluation de la conjonction de ces contraintes]
evaluateBySkill : (*constraintset* : set(Formula)) → (Boolean)
  list($\text{set}_{\text{static}}$(formule), $\text{set}_{\text{dynamic}}$(formule)) ← classify(set(formule), evalScope)
  pour chaque contrainte ∈ $\text{set}_{\text{static}}$(formule) **faire**
    result $\overset{dynamic}{\leftarrow}$ evaluate(contrainte)
    si (result = **FAUX**) **alors** **retourner** HandleException(contrainte)
  pour chaque contrainte ∈ $\text{set}_{\text{static}}$(formule) **faire**
    result $\overset{static}{\leftarrow}$ evaluate(contrainte)
    si (result = **FAUX**) **alors** **retourner** HandleException(contrainte)
retourner **VRAI**

ALG. 9: Processus d'évaluation des contraintes spécialisées qui ont été regroupées selon le critère *evalSkill*

Tout d'abord, il faut déterminer la priorité de l'évaluation de ces contraintes ; cela est fait en se basant sur les propriétés correspondant au critère *evalScope* (voir la section 4.2.2). Ces propriétés (*static* : Formula et *dynamic* : Formula) précisent la construction de la portée de l'évaluation de contrainte. Par défaut, les contraintes qui sont étiquetées par dynamic sont mises en priorité. cela implique la mise à jour des paramètres des contraintes au cours de leur évaluation. Ensuite, les contraintes qui sont étiquetées par static sont évaluées dans leurs portées statiques.

L'évaluation à *VRAI* d'une contrainte spécialisée déclenche l'évaluation de la contrainte spécialisée suivante. L'évaluation à *FAUX* d'une contrainte spécialisée déclenche le traitement d'exception. Les résultats d'évaluation des ces contraintes sont combinés pour produire le résultat final.

4.5 Évaluation d'une contrainte locale

Cette évaluation consiste à évaluer les contraintes spécialisées qui sont regroupées selon le critère *evalPosition*. Ces contraintes spécialisées sont étiquetées par un même tuple de trois propriétés de positionnement. Les noms de chaque propriété appartiennent respectivement au domaine LocationInfo, TimeInfo, EventInfo. Ces contraintes spécialisées font partie de différentes

contraintes globales du plan de coordination sécurisée d'activités ; elles doivent être satisfaites avant ou après l'évaluation de l'appel d'une activité. L'évaluation d'une telle contrainte locale consiste en l'évaluation de la *conjonction* de résultats de chacune de ses contraintes spécialisées correspondantes. Une contrainte locale est dite évaluable par notre algorithme d'évaluation (voir la section 4.3) si et seulement si :

- les contraintes spécialisées correspondant à cette contrainte globale sont toutes évaluables ;
- ces contraintes spécialisées sont toutes évaluées dans le cadre d'une activité spécifique, soit avant , soit après l'évaluation de l'appel correspondant.

*[Input : **set(contrainte)**, un ensemble des contraintes spécialisées à évaluer conjointement]*
[Output : résultat final de l'évaluation de la conjonction de ces contraintes]
evaluateByPosition : (*constraintset* : set(Formula)) → (Boolean)
si (set(contrainte) = ϕ) **alors** **retourner** **VRAI**
pour chaque contrainte ∈ set(contrainte) **faire** *[déterminer les compétences des contraintes]*
set(compétence) ← set(compétence) ∪ (compétence $\overset{evalSkill}{\leftarrow}$ getInfo(contrainte))
[Mettre en priorité des contraintes en se basant sur leur compétence]
set(tuple(priorité, compétence)) ← skillprioritysetting(set(compétence))
pour tuple(priorité, compétence) ∈ set(tuple(priorité, compétence)) **de** la priorité la plus haute **à** la priorité la plus base **faire**
[Éliminer des contraintes redondantes et incohérentes]
$set_{\text{compétence}}$(contrainte) ← redundancyFiltering($set_{\text{compétence}}$(contrainte))
$set_{\text{compétence}}$(contrainte) ← incoherenceFiltering($set_{\text{compétence}}$(contrainte))
[Évaluer des contraintes de chaque aspect]
si ((result ← evaluateBySkill($set_{\text{compétence}}$(contrainte))) = **FAUX**) **alors**
retourner **FAUX**
[Produire le résultat final de l'évaluation]
retourner **VRAI**

Alg. 10: Processus d'évaluation des contraintes correspondant à une contrainte locale

L'algorithme Alg. 10 décrit l'évaluation d'une contrainte locale qui se compose des contraintes spécialisées. Il faut d'abord déterminer les compétences des contraintes spécialisées en fonction du critère *evalSkill* (voir la section 4.2.2) ; cela permet de regrouper les contraintes spécialisées portant sur le même aspect. Ensuite, il faut déterminer leur priorité d'évaluation en fonction du critère *evalPriority* (voir la section 4.2.2). Selon les priorités d'évaluation déterminées, pour chaque aspect, il faut éliminer les contraintes spécialisées redondantes et incohérentes avant de les évaluer par l'algorithme Alg. 9. Finalement, le résultat d'évaluation de contrainte locale est produit en fonction des résultats d'évaluation des ensembles des contraintes de même aspect.

4.6 Évaluation d'une contrainte globale

Une contrainte globale est représentée par une formule logique bien formée. Cette dernière se compose de contraintes spécialisées, qui sont, quant à elle, définies à partir des prédicats prédéfinis pour l'aspect considéré. Les opérateurs d'une contrainte globale ne sont que des connecteurs. La portée d'une contrainte globale se compose de valeurs représentant les résultats d'évaluation (paramètres de sortie) des contraintes constituantes. Le domaine booléen spécifie des valeurs potentielles à affecter à ces paramètres.

L'évaluation d'une contrainte globale revient à vérifier si l'instanciation de ses paramètres

est satisfaite ou non. Les valeurs que les paramètres prennent correspondent aux résultats de l'évaluation de ses contraintes spécialisées constituantes. Néanmoins, l'affectation de valeurs aux paramètres n'est pas faite en même temps : un paramètre ne prend une valeur qu'après l'évaluation de la contrainte spécialisée correspondante. Pour cette raison, une contrainte globale n'est totalement évaluée qu'après avoir évalué toute contrainte spécialisée constituante.

Nous avons besoins d'évaluer partiellement une contrainte globale ; cela permet de détecter si la non-satisfaction d'une contrainte spécialisée implique ou non la non-satisfaction de la contrainte globale correspondante. L'évaluation partielle d'une contrainte globale est décrite comme suit. Initialement, supposons que toutes les contraintes spécialisées constituant la contrainte globale soient évaluées à *VRAI* , alors tous les paramètres de la contrainte globale prennent la valeur *VRAI* . Lors de l'exécution du plan, les valeurs de ces paramètres sont mises à jour après l'obtention du résultat de l'évaluation de la contrainte spécialisée correspondante. Si une contrainte spécialisée est évaluée à *FAUX* , la contrainte globale est réévaluée pour assurer qu'elle est encore satisfaite.

Une contrainte globale est dite évaluable par notre algorithme d'évaluation (voir la section 4.3) si et seulement si :

- la formule représentant cette contrainte globale est bien formée ;
- les contraintes spécialisées correspondant à cette contrainte globale sont toutes évaluables ;
- chacune de ces contraintes spécialisées est à évaluer dans le cadre d'une activité spécifique du plan de coordination sécurisée.

De cette façon, les contraintes globales (de coordination, de sécurité) d'un plan sont contrôlées à travers l'évaluation des contraintes spécialisées dans le cadre de l'exécution des activités.

Exemple. Le plan illustré dans la figure 4.6 est transformé en un plan exécutable comme suit :

1. Les propriétés d'enchaînement sont ajoutées aux spécifications de ces activités en utilisant l'algorithme ALG. 1 comme suit :
   - A1 :
     predecessors : ϕ
     successors : set(ref(A2))
   - A2 :
     predecessors : set(ref(A1))
     successors : set(ref(A3a))
   - A3a :
     predecessors : set(ref(A2))
     successors : set(ref(A3))
   - A3 :
     predecessors : set(ref(A3a))
     successors : set(ref(A4), ref(A9))
   - A4 :
     predecessors : set(ref(A3))
     successors : set(ref(A5), ref(A8))
   - A5 :
     predecessors : set(ref(A4))
     successors : set(ref(A6))
   - A6 :

predecessors : set(ref(A5))

successors : set(ref(A7))

– A7 :

predecessors : set(ref(A6))

successors : set(ref(A9))

– A8 :

predecessors : set(ref(A3))

successors : set(ref(A9))

– A9 :

predecessors : set(ref(A3), ref(A7), ref(A8))

successors : ϕ

2. En cas de changement par rapport à la valeur par défaut, les propriétés d'évaluation correspondant aux critères *evalSkill*, *evalScope*, *evalPosition* (voir la section 4.2.2), *evalStructure*, *evalSatisfaction* (voir la section 4.2.3), *evalPriority* (voir la section 4.2.4) sont ajoutées. Dans ce cas, nous redéfinissons le mode d'évaluation des contraintes spécialisées d'inter-dépendance de données. Elles seront évaluées sur la portée dynamique.

– A2 :

dynamic : matches(A2(*input*(*fullflightdesc*())), A1(*output*(*availableflightlist*())))

– A3a :

dynamic : matches(A3a(*input*(*chosenflightdesc*())), A2(*output*(*chosenflightdesc*())))

– A3 :

dynamic : matches(A3(*input*(*chosenflightdesc*())), A2(*output*(*chosenflightdesc*())))

dynamic : matches(A3(*input*(*clientRef*())), A3(*output*(*clientRef*())))

– A4 :

dynamic : belongs(A4(*input*(*amount*())), A3(*output*(*bookingRef*())))

– A5 :

dynamic : belongs(A5(*input*(*amount*())), A3(*output*(*bookingRef*())))

– A6 :

dynamic : belongs(A6(*input*(*amount*())), A3(*output*(*bookingRef*())))

– A7 :

dynamic : matches(A7(*input*(*bookingRef*())), A3(*output*(*bookingRef*())))

dynamic : matches(A7(*input*(*paymentRef*())), A5(*output*(*paymentRef*())))

– A8 :

dynamic : matches(A8(*input*(*clientRef*())), A3(*input*(*clientRef*())))

dynamic : matches(A8(*input*(*bookingRef*())), A3(*output*(*bookingRef*())))

4.7 Adaptation de la coordination sécurisée

L'exécution d'un plan de coordination sécurisée d'activités consiste à évaluer ses contraintes. Elle repose sur des évaluateurs. Il est souhaitable que cette exécution s'adapte aux nouveaux besoins ou aux changements (de propriétés, de contraintes, d'évaluateurs, etc.) intervenant pendant l'exécution elle-même. L'adaptation consiste en un ensemble d'instructions pour :

– prévoir (au moins avant le démarrage de l'exécution d'un évaluateur) les différentes capacités d'évaluation d'une contrainte pour les évaluateurs.
– choisir les capacités appropriées parmi celles prévues pour les évaluateurs spécifiques ;

– faire des évaluateurs disposant des capacités choisies (au moment de la spécification, de l'interprétation ou de l'exécution) de manière déclarative, interactive et configurable.

La définition de telles instructions sous forme de contraintes et des propriétés de l'aspect adaptation permet de rendre adaptative l'exécution du plan de coordination sécurisée d'activités. Ces propriétés et contraintes, quant à elles, sont définies dans le plan, générées à partir du plan ou renseignées lors de l'exécution du plan. De cette façon, l'adaptation peut être autorisée respectivement au moment de la définition du plan, au moment de la transformation du plan ou au cours de l'exécution du plan. L'aspect adaptation est géré de la même façon que les autres aspects considérés, c.-à-d. par la satisfaction des contraintes et par le traitement d'exception. En premier lieu, cette section identifie les propriétés d'adaptation conformément aux différents critères de l'adaptation. Elle décrit ensuite comment utiliser ces propriétés, ainsi que les prédicats de l'aspect adaptation pour l'exécution adaptative du plan.

4.7.1 Propriétés d'adaptation

Un critère d'adaptation correspond à différents types de propriété ; chaque type de propriété d'adaptation signifie une option de l'évaluation de contrainte (possibilité d'adaptation). Le type de propriété marqué $\natural$ signifie donc l'option par défaut pour cette évaluation. Les quatre critères suivants sont considérés :

adaptOpportunity : Ce critère correspond aux propriétés d'opportunités de l'adaptation qui sont caractérisées de la manière suivante :

$$\mathsf{Property} :\!- \mathsf{tuple}(\mathit{propertyName} : \mathit{OpportunityInfo}, \mathit{propertyValue} : \mathsf{Formula}) \tag{4.7.1}$$

où le domaine $\mathsf{OpportunityInfo} = \{\mathsf{specification}^{\natural}, \mathsf{rendering}, \mathsf{execution}\}$ se compose de valeurs identifiant les phases possibles où l'évaluation d'une contrainte est rendue adaptative. La valeur specification signifie que l'adaptation est autorisée lors de la spécification du plan. La valeur rendering signifie que l'adaptation est autorisée lors de la transformation du plan spécifié en un plan exécutable. La valeur execution signifie que l'adaptation est autorisée lors de l'exécution du plan.

Les propriétés de ce type sont utilisées pour prévoir des moments d'adaptation pour l'évaluation d'une contrainte.

adaptSource : Ce critère correspond aux propriétés de sources de l'adaptation qui sont caractérisées de la manière suivante :

$$\mathsf{Property} :\!- \mathsf{tuple}(\mathit{propertyName} : \mathit{SourceInfo}, \mathit{propertyValue} : \mathsf{choice}(\mathsf{Service}, \mathsf{Operation})) \tag{4.7.2}$$

où le domaine $\mathsf{SourceInfo} = \{\mathsf{internalactor}^{\natural}, \mathsf{externalactor}\}$ caractérise les acteurs qui interviennent à l'exécution d'un plan : il s'agit d'une exécution qui est adaptative par un contrôle interne (par exemple par un moteur de coordination) ou par un contrôle externe (par exemple par un utilisateur qui intervient).

Les propriétés de ce type sont utiles pour prévoir des acteurs d'adaptation pour l'évaluation d'une contrainte.

adaptSubject : Ce critère correspond aux propriétés de raisons de l'adaptation qui sont caractérisées par :

$$\text{Property} :- \text{tuple}(\textit{propertyName} : \textit{SourceInfo}, \textit{propertyValue} : \text{choice}(\text{Service}, \text{Operation})) \quad (4.7.3)$$

où le domaine $\text{subjectInfo} = \{\text{expression}, \text{algorithm}, \text{evaluator}^{\natural}\}$ représente les raisons de l'exécution adaptative du plan. Il s'agit de l'adaptation d'une contrainte du plan (choix parmi des instances de type **Formula**), d'un algorithme d'évaluation d'une contrainte du plan (choix parmi des instances de type **Operation**) ou d'un évaluateur utilisé pour évaluer une contrainte du plan (choix parmi des instances de type **Service**). Les propriétés de ce type sont utiles pour prévoir des raisons d'adaptation pour l'évaluation d'une contrainte.

adaptMechanism : Ce critère correspond aux propriétés de mécanismes de l'adaptation qui sont caractérisées par :

$$\text{Property} :- \text{tuple}(\textit{propertyName} : \textit{MechanismInfo}, \textit{propertyValue} : \text{choice}(\text{Operation}, \text{Service})) \quad (4.7.4)$$

où le domaine $\text{MechanismInfo} = \{\text{interactive}, \text{declarative}^{\natural}, \text{configured}\}$ représente les mécanismes qui rendent adaptative l'exécution du plan : il s'agit de l'adaptation par le biais d'interaction, de déclaration ou de configuration. Les propriétés de ce type sont utiles pour choisir des mécanismes d'adaptation pour l'évaluation d'une contrainte.

4.7.1a Pour l'adaptation lors de la spécification

Les propriétés servant à exprimer l'adaptation lors de la spécification du plan de coordination sécurisée d'activités sont regroupées selon les quatre critères suivants : **necessary**, **linking**, **record** et **visibility**. Ces propriétés sont définies au moment de la spécification du plan. Elles sont utilisées lors de la transformation du plan en un plan exécutable.

necessary : ce critère correspond aux propriétés caractérisées par :

$$\text{Property} :- \text{tuple}(\textit{propertyName} : \textit{NecessaryInfo}, \textit{propertyValue} : \text{choice}(\text{Operation}, \text{Property}, \text{Formula})) \quad (4.7.5)$$

où le domaine $\text{NecessaryInfo} = \{\text{provided}^{\natural}, \text{required}\}$ définit des valeurs qui indiquent la nécessité ou non d'une information définissant le plan de coordination. Il s'agit de contraintes, propriétés ou fonctions fournies et requises au sein des activités du plan.

Les propriétés de ce type sont utiles pour compléter la spécification d'un plan exécutable.

linking : ce critère correspond aux propriétés caractérisées par :

$$\text{Property} :- \text{tuple}(\textit{propertyName} : \textit{LinkingInfo}, \textit{propertyValue} : \text{choice}(\textit{data} : \text{T}, \text{Property})) \quad (4.7.6)$$

où le domaine $\text{LinkingInfo} = \{\text{obligatory}^{\natural}, \text{optional}\}$ définit des valeurs qui indiquent l'obligation ou non de la présence d'une information (propriété ou donnée) définissant un plan. La valeur obligatoire permet d'étiqueter des informations dont la présence dans la spécification d'une activité est obligatoire. La valeur current permet d'étiqueter des propriétés dont la présence dans la spécification est optionnelle. Les propriétés de ce type sont utiles pour le contrôle de l'affectation des valeurs aux paramètres des contraintes à évaluer au cours de l'exécution.

record : ce critère correspond aux propriétés caractérisées par :

$$\mathsf{Property} :\!- \mathsf{tuple}(\mathit{propertyName} : \mathit{RecordInfo}, \mathit{propertyValue} : \mathsf{T}) \tag{4.7.7}$$

où le domaine $\mathsf{RecordInfo} = \{\mathsf{fulltext}, \mathsf{indexing}, \mathsf{encoding}^\natural\}$ caractérise les modes d'enregistrement des informations d'un plan. Il s'agit de données qui sont enregistrées sous forme de données brutes, d'une structure d'indexation de données, ou de données chiffrées.

Les propriétés de ce type facilitent l'exploitation de ces informations. Elles sont utilisées pour choisir des services appropriés qui sont capables d'exploiter ces informations.

Exemple. *fulltext* : (*activityId* : A2) signifie que la valeur A2 représente le nom d'une activité et que cette information est enregistrée en brut.

visibility : ce critère correspond aux propriétés de niveaux de visibilités des informations définissant la coordination sécurisée ; ces propriétés sont caractérisées par :

$$\mathsf{Property} :\!- \mathsf{tuple}(\mathit{propertyName} : \mathit{VisibilityInfo}, \mathit{propertyValue} : \mathsf{T}) \tag{4.7.8}$$

où le domaine $\mathsf{VisibilityInfo} = \{\mathsf{confidential}, \mathsf{private}, \mathsf{public}^\natural\}$ représent e les possibilités de rendre visible une information auprès son exploiteur. La valeur confidential permet d'étiqueter une information comme information privée, elle signifie que cette information requiert un contrôle d'accès. La valeur private signifie que cette information doit être totalement cachée, et la valeur public signifie que cette information ne requiert pas de contrôle d'accès. Les propriétés de ce type sont utilisées pour choisir des services appropriés qui sont capables d'exploiter ces informations.

4.7.1b Pour l'adaptation lors de l'exécution

Il s'agit de cinq critères qui sont définis pour la configuration des activités, des services, ainsi que des journaux de coordination.

observation : ce critère correspond aux propriétés caractérisées par :

$$\mathsf{Property} :\!- \mathsf{tuple}(\mathit{propertyName} : \mathit{ObservationInfo}, \mathit{propertyValue} : \mathsf{T}) \tag{4.7.9}$$

où le domaine $\mathsf{ObservationInfo} = \{\mathsf{previous}^\natural, \mathsf{current}\}$ représente les différents modes d'observation de la valeur effective d'une information. La valeur previous signifie que l'information est observée à la fin de l'exécution de l'activité concernée (phase terminaison). La valeur current signifie que l'information est observée lors de l'exécution de l'activité concernée (phase traitement).

Les propriétés de ce type sont utiles pour le contrôle de l'affectation des valeurs aux paramètres des contraintes qui sont évaluées au cours de l'exécution.

capture : ce critère correspond aux propriétés caractérisées par :

$$\mathsf{Property} :\!- \mathsf{tuple}(\mathit{propertyName} : \mathit{CaptureInfo}, \mathit{propertyValue} : \mathsf{T}) \tag{4.7.10}$$

où le domaine $\mathsf{CaptureInfo} = \{\mathsf{predicted}^\natural, \mathsf{logged}, \mathsf{runtime}\}$ se compose de valeurs représentant les différents modes de capture de la valeur effective d'une information. La valeur defined signifie que la valeur effective d'une information est la valeur par défaut, capturée à partir de la spécification

de l'activité concernée. La valeur logged signifie que la valeur effective d'une information est la valeur enregistrée, capturée à partir du journal de coordination de l'activité concernée. La valeur runtime signifie que la valeur effective d'une information est une valeur observable, capturée au cours de l'exécution de l'activité concernée. Toute propriété de ce type est utile pour l'affectation des valeurs aux paramètres des contraintes à évaluer au cours de l'exécution.

activation : ce critère correspond aux propriétés caractérisées par :

$$\begin{array}{l}\mathsf{Property} :\!- \mathsf{tuple}(\\ \quad \mathit{propertyName} : \mathsf{ActivationInfo},\\ \quad \mathit{propertyValue} : \mathsf{choice}(\mathsf{Service}, \mathsf{Operation}, \mathsf{Activity}, \mathsf{Invocation})\\ )\end{array} \tag{4.7.11}$$

où le domaine $\mathsf{ActivationInfo} = \{\mathsf{activated}, \mathsf{desactivated}^{\natural}\}$ définit des valeurs qui précisent l'usage des services utilisés. Il s'agit des services dont le fonctionnement est activé ou désactivé dans une configuration spécifique.

Les propriétés de ce type sont utiles pour le contrôle de configuration des éléments constitutifs d'une architecture. Ces éléments exécutent le plan de coordination sécurisée d'activités.

configuration : ce critère correspond aux propriétés caractérisées par :

$$\begin{array}{l}\mathsf{Property} :\!- \mathsf{tuple}(\\ \quad \mathit{propertyName} : \mathit{configuration} : \mathsf{ConfigurationInfo},\\ \quad \mathit{propertyValue} : \mathsf{choice}(\mathsf{Service}, \mathsf{Activity}, \mathsf{Property}, \mathsf{Trace}, \mathsf{CoordinationLog})\\ )\end{array} \tag{4.7.12}$$

où le domaine $\mathsf{ConfigurationInfo} = \{\mathsf{basic}^{\natural}, \mathsf{extended}\} \subset \mathsf{dom}(\mathsf{String})$ définit des valeurs qui précisent la configuration d'une architecture de coordination ou la configuration d'un journal de coordination. Cette information a une structure de base ou une structure étendue.

construction : ce critère correspond aux propriétés caractérisées par :

$$\mathsf{Property} :\!- \mathsf{tuple}(\mathit{propertyName} : \mathit{ConstructionInfo}, \mathit{propertyValue} : \mathsf{Service})) \tag{4.7.13}$$

où le domaine $\mathsf{ConstructionInfo} = \{\mathsf{alone}, \mathsf{wrapped}, \mathsf{coordinated}\}$ définit des valeurs précisent la construction d'une architecture correspondant à un service qui exécute le plan de coordination. Il s'agit de la mise en œuvre selon une des trois approches : version sécurisée, configuration sécurisée et parcellaire.

Les propriétés de ce type sont utiles pour la génération des propriétés par défaut correspondant à une construction.

4.7.2 Contraintes d'adaptation

Étant donné

– $\mathsf{FormulaProperty} \subset \mathsf{Property}$, le domaine des propriétés pouvant étiqueter les contraintes. Il s'agit des propriétés dont la valeur est de type Formula.

- $\mathsf{T_1} :- \mathsf{choice}(\mathsf{Formula}, \mathsf{tuple}(\mathsf{Formula}, \mathsf{set}(\mathsf{FormulaProperty}))) \in \nabla$, le type qui caractérise les contraintes dont il est souhaitable d'adapter l'évaluation. Il s'agit d'une formule typique (c.-à-d. une instance du type Formula) ou d'une formule associée à des propriétés spécifiques (c.-à-d. une instance du type construit $\mathsf{tuple}(\mathsf{Formula}, \mathsf{set}(\mathsf{FormulaProperty}))$).
- le type $\mathsf{T_2}$ qui caractérise les éléments intervenant dans l'adaptation. Il s'agit de propriété, contrainte, algorithme, service, activité, appel (typiques ou avec des propriétés spécifiques) servant à leur adaptation :

$$\begin{aligned}\mathsf{T_2} :- \mathsf{choice}(&\mathsf{Property},\\ &\mathsf{Formula}, \mathsf{tuple}(\mathsf{Formula}, \mathsf{set}(\mathsf{Property})),\\ &\mathsf{Operation}, \mathsf{tuple}(\mathsf{Operation}, \mathsf{set}(\mathsf{Property})),\\ &\mathsf{Service}, \mathsf{tuple}(\mathsf{Service}, \mathsf{set}(\mathsf{Property})),\\ &\mathsf{Activity}, \mathsf{tuple}(\mathsf{Activity}, \mathsf{set}(\mathsf{Property})),\\ &\mathsf{Invocation}, \mathsf{tuple}(\mathsf{Invocation}, \mathsf{set}(\mathsf{Property})))\end{aligned} \in \nabla$$

Les prédicats suivants sont définis pour l'aspect adaptation :

1. $\mathtt{predictive}(\mathsf{T_1}(), \mathsf{T_2}()) \in \mathcal{P}$: ce prédicat permet de prévoir une capacité d'évaluation (possibilité d'adaptation) pour un évaluateur ;
2. $\mathtt{contractual}(\mathsf{T_1}(), \mathsf{T_2}()) \in \mathcal{P}$: ce prédicat permet de choisir automatiquement une adaptation pour un évaluateur, en gardant la possibilité d'une intervention au cours de l'exécution ;
3. $\mathtt{assigned}(\mathsf{T_1}(), \mathsf{T_2}()) \in \mathcal{P}$: ce prédicat permet à un évaluateur de disposer d'une capacité d'évaluation ;

Ces prédicats sont utilisés pour formuler des contraintes dans plusieurs cas d'adaptation au niveau de la spécification ainsi qu'au niveau de l'exécution d'un plan de coordination sécurisée.

4.7.2a Adaptation au niveau de la spécification du plan de coordination sécurisée d'activités

A ce niveau, il existe différentes instructions pour transformer une contrainte (de coordination, de sécurité). Ces instructions sont exprimées par des contraintes d'adaptation, et ces dernières sont basées sur des prédicats définis comme suit :

Instructions permettant de redéfinir les contraintes : étant donné
$\mathsf{T} :- \mathsf{choice}(\mathsf{Operation}, \mathsf{Service}) \in \nabla$, le type qui caractérise les algorithmes d'évaluation sémantique et syntaxique des contraintes, ou les services qui les implantent, les prédicats suivants définissent ce type d'instructions :

- $\mathtt{predictive}(\mathtt{Formula}(\mathsf{f}), \mathtt{tuple}(\mathtt{set}(\mathtt{Property}(\mathsf{p})), \mathtt{T}(\mathsf{t})))$: spécifie que l'algorithme t est utilisé pour évaluer la formule f ; les propriétés p de f sont utilisées comme des paramètres d'entrée de cet algorithme pour cette évaluation.
- $\mathtt{contractual}(\mathtt{Formula}(\mathsf{f}), \mathtt{tuple}(\mathtt{set}(\mathtt{Property}(\mathsf{p})), \mathtt{T}(\mathsf{t})))$: spécifie que l'algorithme t est utilisé pour évaluer la formule f ; les propriétés p de f sont utilisées comme des paramètres d'entrée de cet algorithme pour cette évaluation.

– `assigned(Formula(f), tuple(set(Property(p)), T(t)))` : spécifie que l'algorithme `t` est utilisé pour évaluer la formule `f` ; les propriétés `p` de `f` sont utilisées comme des paramètres d'entrée de cet algorithme pour cette évaluation.

Exemple. Soit coherent : (*srcFormula* : Formula) → (*result* : Boolean), une fonction du type Formula. Elle permet de vérifier la cohérence des prédicats constituant une contrainte. Étant donnée une formule, cette fonction rend *VRAI* si les prédicats la constituant sont cohérents, *FAUX* sinon.

`contractual(Formula(f), tuple(?, Operation(coherent : (srcFormula : Formula) → (result : Boolean))))` est une contrainte spécialisée. Elle spécifie que les prédicats constituant la formule `f` doivent être cohérents, conformément à un algorithme de vérification de la cohérence (représenté par la fonction coherent). Notons qu'elle ne spécifie pas le service qui fournit cette fonction.

`contractual(Formula(f), tuple(?, Operation(antiredundant : (srcFormula : Formula) → (dstFormula : Formula))))` est une contrainte spécialisée d'adaptation. Elle spécifie que les prédicats constituant la formule `f` doivent être non redondants, conformément à un algorithme de vérification de la redondance (représenté par la fonction antiredundant du type Formula). Notons que cette contrainte ne spécifie pas le service qui fournit cette fonction.

Instructions permettant d'optimiser les contraintes : Étant donné
T :– choice(Property, Formula, Operation) $\in \nabla$, le type qui caractérise les contrats sémantiques de l'évaluation des contraintes, les prédicats prédéfinis définissent les instructions qui suivent :

– `predictive(Formula(f), T(t))` : spécifie la prédiction de relation entre une contrainte `f` et une propriété, une formule ou une opération `t`.
– `contractual(Formula(f), T(t))` : spécifie le choix de relation explicite entre une contrainte `f` et une propriété, une formule ou une opération `t`.
– `assigned(Formula(f), T(t))` : spécifie la relation explicite entre une contrainte `f` et une propriété, une formule ou une opération `t`.

Exemple. Soit substitute : (*srcFormula* : Formula) → (*dstFormula* : Formula), une fonction du type Formula définissant la règle de substitution. Étant donnée une formule, cette fonction va rendre une formule substitutive si elle existe. Sinon elle rendra la formule initiale.

`contractual(Formula(f), Operation(substitute))` est une contrainte spécialisée d'adaptation. Elle définit une instruction de la substitution d'une contrainte `f` en fonction du prédicat `contractual()` $\in \mathcal{P}$. Cette contrainte spécifie que `f` peut être substituée par une contrainte équivalente en utilisant l'algorithme de substitution (représenté par la fonction substitute du type Formula). Notons que ce prédicat ne spécifie pas le service qui fournit cette fonction.

`assigned(Formula(f), tuple(?, Service(`*serviceName*`(compiler1), `*operationName*`(equivalent))))` est une autre contrainte spécialisée d'adaptation. Elle spécifie que les prédicats constituant la formule `f` doivent être remplacés par des prédicats équivalents conformément à un algorithme de vérification de l'équivalence. La fonction equivalent : (*srcFormula* : Formula) → (*dstFormula* : Formula) du type Formula implante cet algorithme. Cette fonction est fournie par le service compiler1.

`predicted(Formula(f), tuple(?, Service(`*serviceName*`(compiler1), `*operationName*`(implicate))))` est une contrainte spécialisée d'adaptation. Elle spécifie que les prédicats constituant la formule `f` doivent être remplacés par des prédicats qui les impliquent conformément à un algorithme de vérification de l'implication. La fonction (du type Formula)
implicate : (*srcFormula* : Formula) → (*dstFormula* : Formula) implante cet algorithme. Cette fonction est fournie par le service de nom compiler1.

4.7.2b Adaptation au niveau de l'exécution du plan de coordination sécurisées des activités

A ce niveau, différentes instructions pour évaluer des contraintes sont définies en termes de contraintes d'évaluation à base de prédicats prédéfinis comme suit :

Instructions pour l'évaluation d'une contrainte locale : ces instructions contrôlent l'évaluation d'un ensemble de contraintes spécialisées à satisfaire conjointement avant ou après un appel. Les contraintes d'adaptation représentant telles instructions portent sur les contraintes spécialisées qu'il est nécessaire d'adapter leur évaluation. Pour l'instant, nous nous appuyons sur l'adaptation de l'évaluation des contraintes de coordination et de sécurité.

• **Mise en priorité des contraintes spécialisées constituant une contrainte locale :** les contraintes d'adaptation de priorité d'évaluation sont définies à partir des prédicats d'adaptation. Elles portent sur les contraintes spécialisées (autres aspects que l'aspect adaptation : coordination, sécurité, etc.) et leur propriété evalPriority (qui permet d'interpréter la priorité d'évaluation de chaque contrainte).

- `predictive(Formula(f), evalPriority(t))` : spécifie la prédiction de niveau de priorité `t` pour la contrainte `f` qui constitue une contrainte locale spécifique.
- `contractual(Formula(f), evalPriority(t))` : spécifie le choix automatique de niveau de priorité `t` pour la contrainte `f` qui constitue une contrainte locale spécifique.
- `assigned(Formula(f), evalPriority(t))` : spécifie la disposition explicite de niveau de priorité `t` pour la contrainte `f` qui constitue une contrainte locale spécifique.

Exemple. `assigned(tuple(Formula(), evalSkill(authentication)), evalPriority(1))` est la contrainte spécialisée de l'aspect adaptation. Elle spécifie que les contraintes spécialisées de l'aspect authentification doivent être évaluées avec la première priorité.

• **Choix du mode d'évaluation :** Les contraintes d'adaptation de priorité d'évaluation sont définies à partir des prédicats d'adaptation. Elles portent sur les contraintes spécialisées des autres aspects (coordination, sécurité, etc.) et leur propriété evalScope. Ces contraintes permettent de préciser le mode d'évaluation d'une contrainte :

- `predictive(Formula(f), evalScope(t))` : spécifie la prédiction de mode d'évaluation de `t` pour la contrainte `f` qui constitue une contrainte locale spécifique.
- `contractual(Formula(f), evalScope(t))` : spécifie le choix automatique du mode d'évaluation de `t` pour la contrainte `f` qui constitue une contrainte locale spécifique.
- `assigned(Formula(f), evalScope(t))` : spécifie la disposition explicite du mode d'évaluation de `t` pour la contrainte `f` qui constitue une contrainte locale spécifique.

Exemple. `contractual(Formula(belongs()), evalScope(dynamic))` est une contrainte globale d'adaptation. Elle spécifie que toute contrainte spécialisée définie à partir du prédicat d'inter-dépendance de données `belongs` sera évaluée sur la portée dynamique.
Considérons les contraintes spécialisées constituant la pré-condition de l'appel dans le cadre de l'activité A4 (voir la figure 3.31). Il s'agit d'une contrainte locale représentant la conjonction des contraintes spécialisées de différents aspects, dont la contrainte d'inter-dépendance de données `belongs(A4(input(amount())), A3(output(bookingRef())))`. Par défaut, ces contraintes spécialisées sont évaluées sur les portées statiques. Avec la présence de la contrainte globale d'adapta-

tion décrite ci-dessus, la contrainte belongs(A4(*input*(*amount*())), A3(*output*(*bookingRef*()))) sera évaluée sur la portée dynamique.

• **Choix de l'évaluateur :**
- predictive(Formula(f), Service(t)) : spécifie la prédiction de contrat architectural de l'évaluation de t pour la contrainte f qui constitue une contrainte locale spécifique.
- contractual(Formula(f), Service(t)) : spécifie le choix automatique de contrat architectural de l'évaluation t pour la contrainte f qui constitue une contrainte locale spécifique.
- assigned(Formula(f), Service(t)) : spécifie la disposition explicite de de contrat architectural de l'évaluation de t pour la contrainte f qui constitue une contrainte locale spécifique.

Exemple. contractual(Formula(belongs()), Service(ref(solver1))) est une contrainte d'adaptation. Elle spécifie que l'évaluation de toute contrainte spécialisée définie à partir du prédicat d'inter-dépendance de données belongs sera exécutée par l'évaluateur solver1.
contractual(Formula(belongs()), Service(ref(solver1)))∨
contractual(Formula(belongs()), Service(ref(solver2)))
est une contrainte d'adaptation autorisant le choix des évaluateurs. L'évaluation de toute contrainte spécialisée définie à partir du prédicat d'inter-dépendance de données belongs sera exécutée par l'évaluateur solver1 ou solver2.
Considérons la pré-condition de l'appel dans le cadre de l'activité A4 (voir la figure 3.31), dont la contrainte de données belongs(A4(*input*(*amount*())), A3(*output*(*bookingRef*()))). L'évaluation de cette dernière sera effectuée par l'évaluateur par défaut ou par les évaluateurs précisés dans la contrainte globale d'adaptation décrite ci-dessus.

Instructions pour l'évaluation d'une contrainte globale : les instructions suivantes portent sur un ensemble de contraintes de même aspect (de coordination, de sécurité, etc.).

• **Choix du niveau de satisfaction :**
- predictive(Formula(f), *evalSatisfaction*(t)) : spécifie la prédiction de niveau de satisfaction t pour la contrainte f qui constitue une contrainte locale spécifique.
- contractual(Formula(f), *evalSatisfaction*(t)) : spécifie le choix automatique de niveau de satisfaction t pour la contrainte f qui constitue une contrainte locale spécifique.
- assigned(Formula(f), *evalSatisfaction*(t)) : spécifie la disposition explicite de niveau de satisfaction t pour la contrainte f qui constitue une contrainte locale spécifique.

• **Choix de l'algorithme :**
- predictive(Formula(f), Operation(t)) : spécifie la prédiction de l'algorithme d'évaluation de t pour la contrainte f qui constitue une contrainte locale spécifique.
- contractual(Formula(f), Operation(t)) : spécifie le choix automatique de l'algorithme d'évaluation de t pour la contrainte f qui constitue une contrainte locale spécifique.
- assigned(Formula(f), Operation(t)) : spécifie la disposition explicite de l'algorithme d'évaluation de t pour la contrainte f qui constitue une contrainte locale spécifique.

Exemple. La contrainte d'adaptation suivante autorise le choix des algorithmes :
contractual(Formula(belongs()), Operation(ref(algo1)))∨
contractual(Formula(belongs()), Service(ref(algo2))).

L'algorithme algo1 et algo2 sont disponibles pour l'évaluation de toute contrainte spécialisée définie à partir du prédicat d'inter-dépendance de données `belongs`.

4.8 Conclusion

Dans ce chapitre, nous avons présenté une approche qui permet d'exécuter un plan de coordination sécurisée d'activités. L'exécution d'un plan revient à l'exécution de ses activités. Cette exécution repose sur des évaluateurs de contraintes du plan. Pour cela, le plan est transformé avec des informations utiles et nécessaires pour son exécution. Ces informations deviennent des propriétés (exploitables) des activités auxquelles les contraintes devant être évaluées sont associées. Cette approche offre un algorithme général permettant l'évaluation des contraintes définies par des formules bien formées. Les différentes versions de cet algorithme autorise l'optimisation de l'évaluation des contraintes spécialisées, locales et globales du plan.

Cette approche permet également de maintenir une projection explicite entre la spécification et la réalisation de différents aspects d'une exécution coordonnée des services. De cette façon, il permet une gestion cohérente et sûre de l'exécution des mesures qui répondent aux besoins sur différents aspects.

L'exécution d'un plan selon cette approche peut s'adapter aux nouveaux besoins et aux changements (de propriétés, de contraintes, d'évaluateurs, etc.) intervenant pendant l'exécution même du plan. L'adaptation peut s'exprimer lors de la spécification, lors de la transformation ou lors de l'exécution du plan. Des mesures adhoc et hétérogènes (modèles, architectures, mécanismes, outils, matériaux, etc.) de sécurité qui ne sont applicables que dans des contextes d'exécution concrets peuvent être préalablement considérées lors de spécification de l'exécution coordonnée de services. Elles sont explicitement définies en terme des contraintes et des propriétés des aspects coordination et sécurité considérés, ou de l'aspect adaptation. De cette façon, l'adaptation peut être respectivement autorisée au moment de la définition du plan, de l'interprétation du plan ou au cours de l'exécution du plan. Cette adaptation est gérée de manière similaire aux autres aspects considérés du plan.

Chapitre 5

MEOBI, canevas de coordination sécurisée

Ce chapitre introduit MEOBI, notre canevas de coordination sécurisée. MEOBI supporte la construction des applications sécurisées à base de services par la voie de la coordination sécurisée des services. Une telle construction comprend la spécification, l'exécution et la gestion d'un plan de coordination sécurisée d'activités. Le plan, quant à lui, est défini conformément au modèle MEO (voir le chapitre 3). L'exécution du plan consiste en l'évaluation des contraintes du plan ; elle utilise des évaluateurs (voir le chapitre 4).
MEOBI est défini sous forme d'un ensemble d'interfaces et de classes abstraites permettant d'instancier les moteurs d'exécution des plans. Les interfaces, ainsi que les classes abstraites qui les implantent partiellement, servent à définir la structure des informations spécifiant les plans. Elles servent également à définir la façon (sécurisée et non sécurisée) dont sont manipulées et exploitées ces informations, en particulier les activités, les contraintes et les journaux de coordination. De même, ces interfaces et classes abstraites définissent la façon dont sont produites les informations utiles pour l'exécution. Elles servent à définir l'architecture de la gestion et de l'exécution sûre et adaptative du plan : les composants et leurs relations pendant l'exécution d'une coordination sécurisée.
Ce chapitre est organisé de la manière suivante. La section 5.1 présente les interfaces définissant les composants de l'architecture générale du canevas. La section 5.2 détaille les composants et leurs interactions permettant d'exécuter une activité. La section 5.3 présente comment la gestion de la sécurité s'articule dans l'exécution d'une activité. La section 5.4 présente comment assurer la sûreté de fonctionnement de l'exécution d'un plan. La section 5.5 présente comment instancier le canevas pour la construction des applications à base de services.

Avant de présenter en détail la spécification du canevas MEOBI, nous voulons rappeler certains points importants de notre approche. Selon MEO (voir le chapitre 3), un service est identifié par un nom ; il exporte une interface et son mode d'emploi. L'interface définit un ensemble de signatures de fonctions. Le mode d'emploi définit les propriétés du service et les contraintes sur les fonctions et les propriétés de ce service.

Une application à base de services est elle-même un service. Elle est construite à partir d'activités. Ces activités correspondent aux appels à des fonctions de services existants. Elles sont contrôlées par un plan de coordination sécurisée. Le plan précise les instructions concernant les aspects coordination et les aspects sécurité. Ces instructions sont définies en termes de contraintes (globales, locales, spécialisées). Tandis que les contraintes spécialisées sont associées aux activités comme pré-condition et post-condition des appels, les contraintes locales et globales sont associées aux activités en tant que propriété de sûreté de fonctionnement.

Le processus d'exécution interne de l'application à base de services correspond donc à l'exécu-

tion d'un plan de coordination sécurisée de ses activités. Cette exécution consiste en l'évaluation des pré-conditions de l'appel, l'évaluation de l'appel et l'évaluation des post-conditions de l'appel (voir le chapitre 4). Elle repose sur des évaluateurs.

Le canevas MEOBI permet de développer des applications à base de services conformément au modèle MEO. Il supporte la spécification et l'exécution des plans de coordination sécurisée d'activités en fonction de ses interfaces et de ses classes abstraites. Ces dernières, quant à elles, se relient à une ou deux interfaces Information, Service (voir la figure 5.1).
Les informations spécifiant un plan (la structure d'une information correspond à un type de ∇, voir le chapitre 3) sont représentées par des interfaces qui héritent de l'interface Information de MEOBI. Comme illustrée dans la figure 5.1, l'interface Information définit certaines fonctions (de n'importe quel type de ∇) qui permettent de manipuler les informations spécifiant un plan. La fonction setInformation permet de définir ou de modifier les valeurs effectives pour des informations. Les fonctions getInformation et extractInformation permettent de récupérer (totalement ou partiellement) les valeurs effectives des informations. La fonction exportAs permet de transformer les valeurs effectives des informations en différents formats, par exemple emballer une information dans un message qui est prêt à communiquer ou chiffrer cette information. La fonction labelInformation permet d'étiqueter les informations spécifiant le plan. Les classes (abstraites),

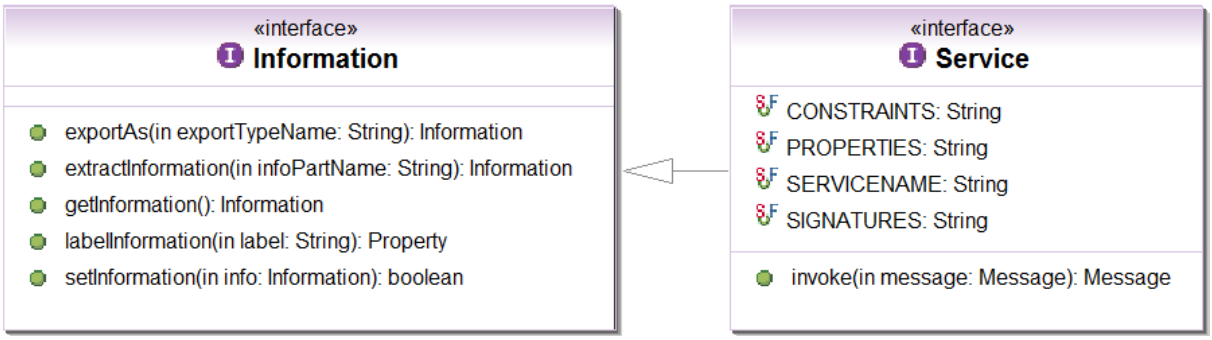


Fig. 5.1 – Interfaces Information et Service

implantant (partiellement) l'interface Information ou bien implantant (partiellement) une interface qui en hérite, permettent de créer les structures des informations définissant le plan comme décrit dans le chapitre 3. Elles permettent également de manipuler et de supprimer ces structures. Par exemple, considérons l'interface Activity qui représente des activités dont la structure est définie dans la section 3.2.4. Cette interface hérite de l'interface Information. Une fois que cette interface est instanciée, la fonction getInformation() permet de récupérer une activité entière. La fonction extractInformation() permet de récupérer le nom, les propriétés, l'appel, les pré-conditions ou les post-conditions d'une activité.
Les composants de l'architecture d'exécution (qui opèrent sur les informations spécifiant le plan) sont représentés par des interfaces qui héritent de l'interface Service. Cette interface hérite elle aussi des fonctions de l'interface Information. Cela permet d'exhiber le nom, les fonctions et le mode d'emploi des composants dont la structure a été décrite dans la section 3.2.2. La séparation de fonctionnement des interfaces qui héritent de Service permet de faciliter les modifications et les adaptations éventuelles des composants. Comme illustrée dans la figure 5.1, l'interface Service définit certaines constantes permettant de rappeler la structure d'un composant. La fonction invoke permet d'appeler les fonctions des composants de l'architecture de manière non sécurisée ou de manière sécurisée (conformément à leurs modes d'emploi).

Une application à base de services est implantée par une instance du canevas, c.-à-d. une configuration particulière des composants de son architecture.

5.1 Architecture du canevas

La figure 5.2 illustre l'architecture du canevas MEOBI. Cette architecture se compose des contrôleurs, des évaluateurs et des services utilisés. Ils interagissent au sein des appels à leurs fonctions.

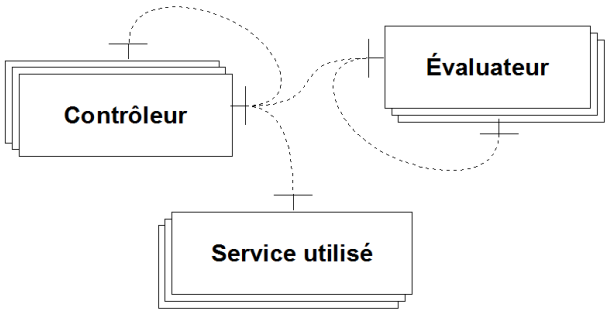


FIG. 5.2 – Architecture du canevas

Un moteur d'exécution d'un plan de coordination sécurisée d'activités correspond à une configuration spécifique des composants de cette architecture. Dans cette configuration, l'interaction entre un service appelant (un composant qui utilise une fonction) et un service appelé (un composant qui fournit une fonction) se passe en quatre temps : l'abonnement, l'activation, l'exécution, la terminaison.

Dans le premier temps, un service appelant s'abonne auprès des contrôleurs ou auprès des services qu'il veut utiliser. Cela correspond à l'instanciation des composants dans la configuration selon les contraintes globales et les propriétés de ce plan. Dans le second temps, une session de communication est créée pour le service appelant abonné, il est donc activé dans le cadre de cette session. Cela correspond à la mise en jeu d'un contrôleur dédié à son interaction. Dans le troisième temps, l'exécution de l'appel de ce service appelant activé est achevée. Cela correspond à l'exécution du contrôleur d'interaction. Dans le quatrième temps, le service appelant reçoit le résultat d'appel qui contient éventuellement le résultat d'exécution de la fonction appelée. La session de communication est fermée.

La suite de cette section présente donc les composants de l'architecture.

5.1.1 Évaluateur

Un évaluateur s'occupe de l'évaluation de contrainte d'un plan. Comme illustré dans la figure 5.3, un évaluateur est représenté par la classe abstraite ConstraintSolver. Cette dernière met en œuvre partiellement l'interface Constraint ou bien une interface qui en hérite. La fonction evaluate correspond à un algorithme d'évaluation de contrainte spécifique, par exemple l'algorithme ALG. 5 présenté dans la section 4.3.1. Les types d'évaluateur suivants (étant sous classes de ConstraintSolver) sont considérés en tant que composants de l'architecture du canevas :

– **Évaluateurs spécialisés** (interface SpecialistConstraint, voir la section 5.2.3) : la fonction evaluate est dédiée à l'évaluation d'une contrainte spécialisée particulière ou les contraintes spécialisées d'un aspect.

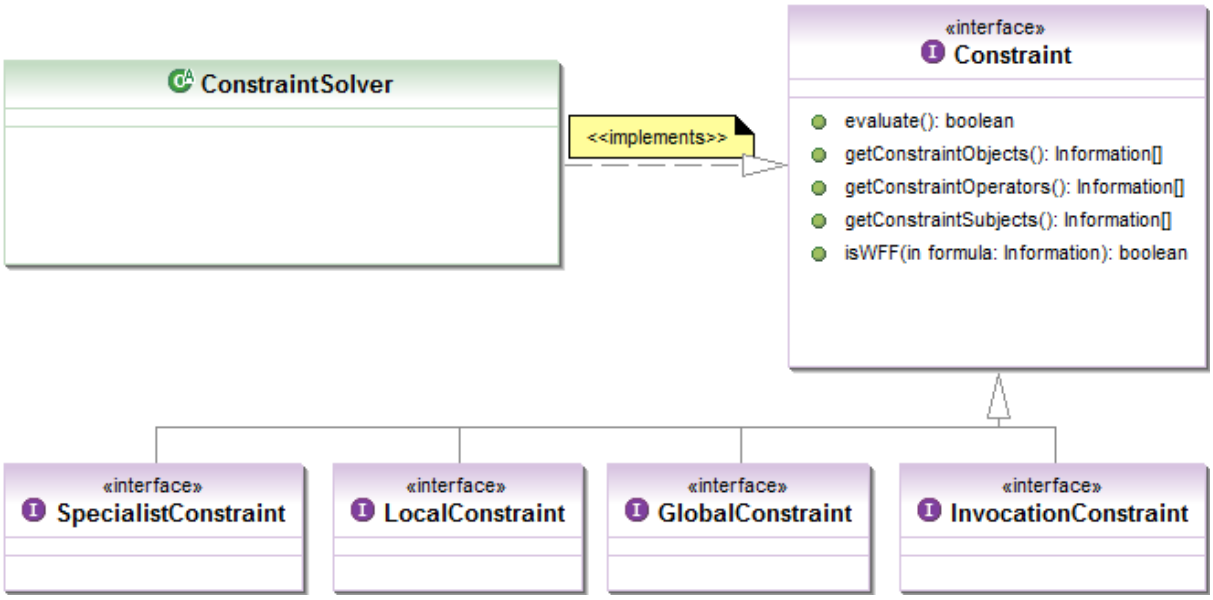


Fig. 5.3 – Diagramme de classes représentant les évaluateurs

- **Évaluateurs d'appel** (interface InvocationConstraint, voir la section 5.2.4) : la fonction evaluate est dédiée à l'évaluation d'une ou d'un groupe d'appel.
- **Évaluateurs locaux** (interface LocalConstraint, voir la section 5.2.5) : la fonction evaluate est dédiée à l'évaluation d'une ou d'un groupe de contraintes locales,
- **Évaluateurs globaux** (interface GlobalConstraint, voir la section 5.4.1) : la fonction evaluate est dédiée à l'évaluation d'une ou d'un groupe de contraintes globales.

5.1.2 Contrôleur

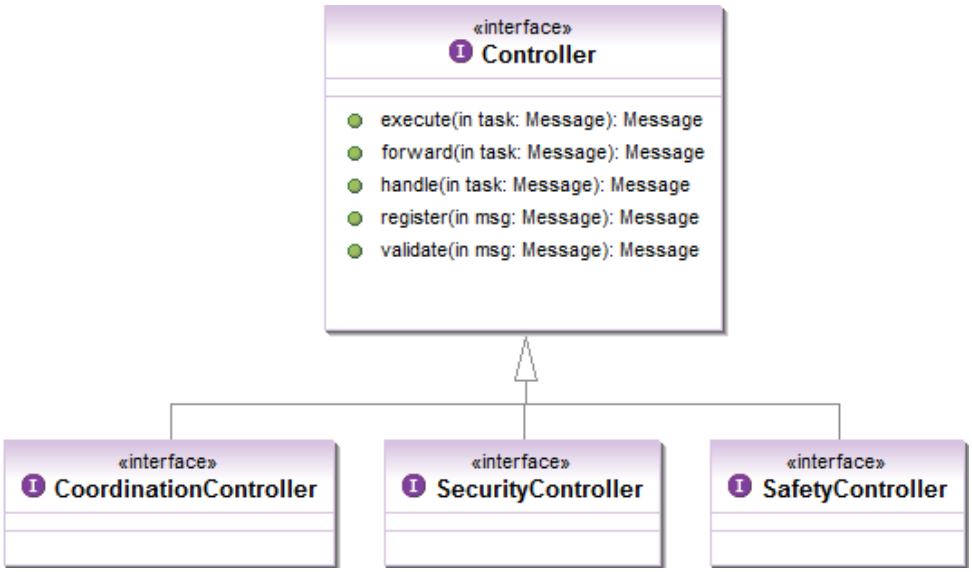


Fig. 5.4 – Diagramme de classes représentant les contrôleurs

Comme illustré dans la figure 5.4, un contrôleur est représenté par l'interface Controller ou une des interfaces qui en hérite. Un contrôleur offre des fonctions de gestion et de contrôle aux composants de l'architecture. La fonction register définie par l'interface Controller permet à

un composant de s'abonner auprès d'un contrôleur. La fonction handle permet à un composant abonné d'adresser une instruction (intervention, notification, etc.) à un contrôleur. Un contrôleur offre également des fonctions qui permettent aux évaluateurs d'achever l'évaluation des contraintes. La fonction execute définie par l'interface Controller permet à un composant abonné de mettre en service un contrôleur. La fonction validate permet de valider un critère de qualité par un mécanisme spécifique à un contrôleur. La fonction forward de Controller permet de faire suivre les fonctionnalités (de contrôle, de validation) aux services utilisés en appelant les fonctions équivalentes de ces derniers.

Les types de contrôleurs suivants (héritant de Controller) sont considérés en tant que composants de l'architecture du canevas MEOBI :

- **Contrôleurs de coordination** (interface CoordinationController) : ils contrôlent les aspects coordination de l'exécution d'un plan. Il y a deux types de contrôleurs de coordination (les interfaces qui les représentent héritent de CoordinationController) :
  - Les coordinateurs, représentés par l'interface Coordinator (voir la section 5.2.1), servent à l'exécution des activités d'un plan.
  - Les contrôleurs d'interaction, représentés par l'interface InteractionController (voir la section 5.2.2), sont dédiés à l'interaction entre les services faisant partie de cette exécution.
- **Contrôleurs de sécurité** (interface SecurityController) : ils contrôlent les aspects sécurité de l'exécution d'un plan (authentification, autorisation, etc.). Ils sont décrits en détail dans la section 5.3.
- **Contrôleurs de sûreté de fonctionnement** (interface SafetyController) : ils s'occupent de la sûreté de fonctionnement des services faisant partie de l'exécution d'un plan. A part les évaluateurs globaux, il y a deux types de contrôleurs de sûreté de fonctionnement (les interfaces qui les représentent héritent de SafetyController) :
  - Les contrôleurs d'exception, représentés par l'interface ExceptionController (voir la section 5.4.2), offrent des mesures de traitement d'exception.
  - Les contrôleurs d'adaptation, représentés par l'interface AdaptationController (voir la section 5.4.3), offrent des mesures permettant de contrôler les changements de l'exécution du plan.

5.1.3 Services utilisés

Les fonctions exportées par ce type de services sont utilisées pour réaliser les fonctionnalités d'une application à base de services. Ce sont :

- les fonctions qui sont appelées dans le cadre des activités d'un plan ; les services qui les fournissent jouent donc le rôle de **services à coordonner** dans le cadre de l'exécution du plan.
- les fonctions qui sont éventuellement appelées pour achever l'exécution d'une activité ou l'évaluation d'une contrainte, par exemple les fonctions des services d'interaction ou de réseaux d'une infrastructure de communication, ou les services génériques de sécurité ; les services qui les fournissent jouent donc le rôle de **tiers de confiance** dans le cadre de l'exécution du plan.

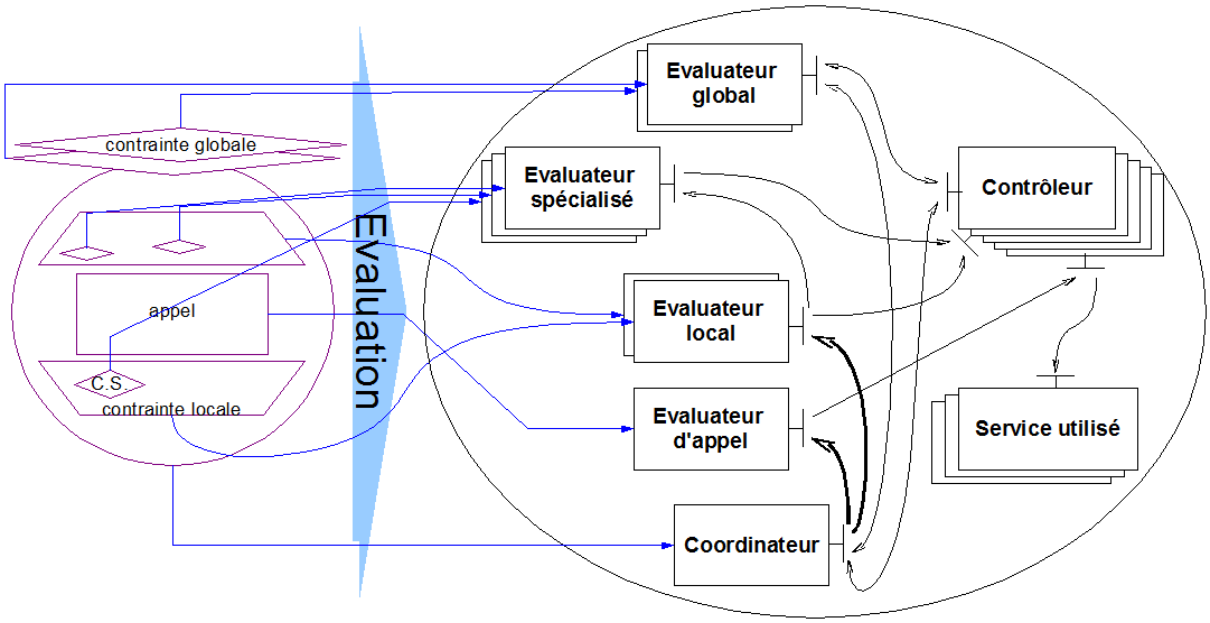


Fig. 5.5 – Architecture d'exécution d'une activité du plan de coordination sécurisée

5.2 Composants pour l'exécution d'une activité

Comme décrit dans la section 4.1, l'exécution d'une activité consiste à l'évaluation de la pré-condition de l'appel, l'évaluation de l'appel à une fonction d'un service (à coordonner) et l'évaluation de sa post-condition. La figure 5.5 illustre la correspondance entre les informations spécifiant une activité et les composants de l'architecture qui l'exécute. Le coordinateur s'occupe du processus d'exécution de l'activité. Les évaluateurs locaux s'occupent des contraintes locales (pré/post-condition) à l'aide des évaluateurs spécialisés. L'évaluateur d'appel s'occupe de l'appel. Les évaluateurs globaux s'occupent des contraintes globales, cela implique des interventions au niveau du processus d'exécution de l'activité. Les contrôleurs contribuent à l'exécution de l'activité et à l'évaluation des contraintes. Cette section détaille comment l'exécution d'une activité est implantée par les composants de notre architecture.

5.2.1 Coordinateur

Un coordinateur est un contrôleur dédié à l'exécution des activités du plan. Il est représenté par l'interface Coordinator. La figure 5.6 illustre les fonctions fournies par un évaluateur. La fonction exec met en œuvre le processus d'exécution d'une activité qui a été décrit dans la section 4.1. Pour cela, le coordinateur fait exécuter l'activité aux évaluateurs d'appel et aux évaluateurs locaux. Ce contrôle est réalisé de manière interactive. Il est conforme aux instructions du plan : quelle contrainte et quand doit elle être satisfaite en rapport avec l'exécution de l'appel concerné.

La figure 5.7 illustre les interactions principales d'un coordinateur. Un coordinateur interprète les contraintes globales qui régissent différents aspects de l'exécution de l'activité dont il s'occupe. En fonction de ces contraintes globales, il crée ou s'abonne aux contrôleurs et aux évaluateurs globaux de ces aspects. Un coordinateur crée ou s'abonne à un contrôleur de non-répudiation pour l'activité dont il s'occupe, même s'il n'existe pas une contrainte globale de non-répudiation.

Au cours de son exécution, le coordinateur reçoit et met en œuvre des instructions venant

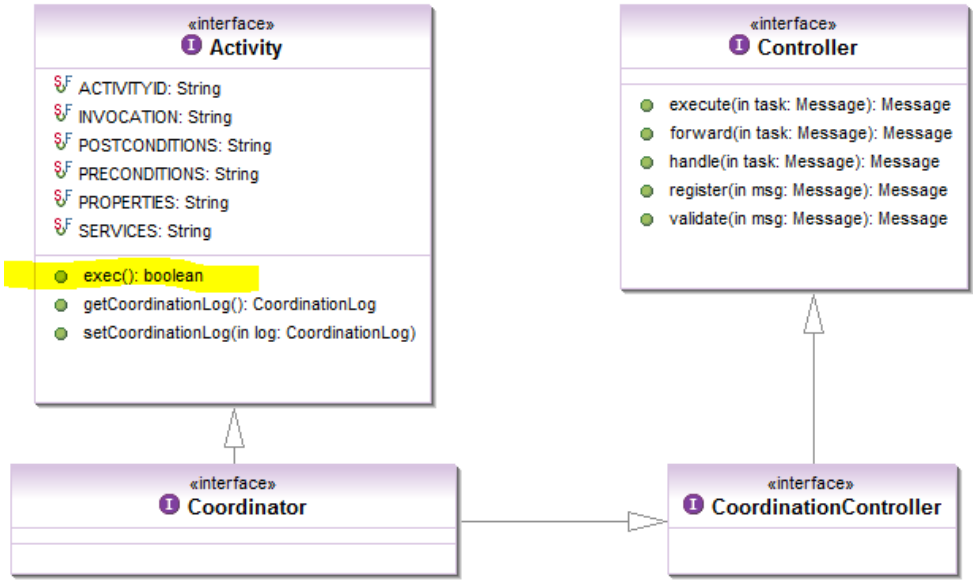


FIG. 5.6 – Diagramme de classes pour les coordinateurs

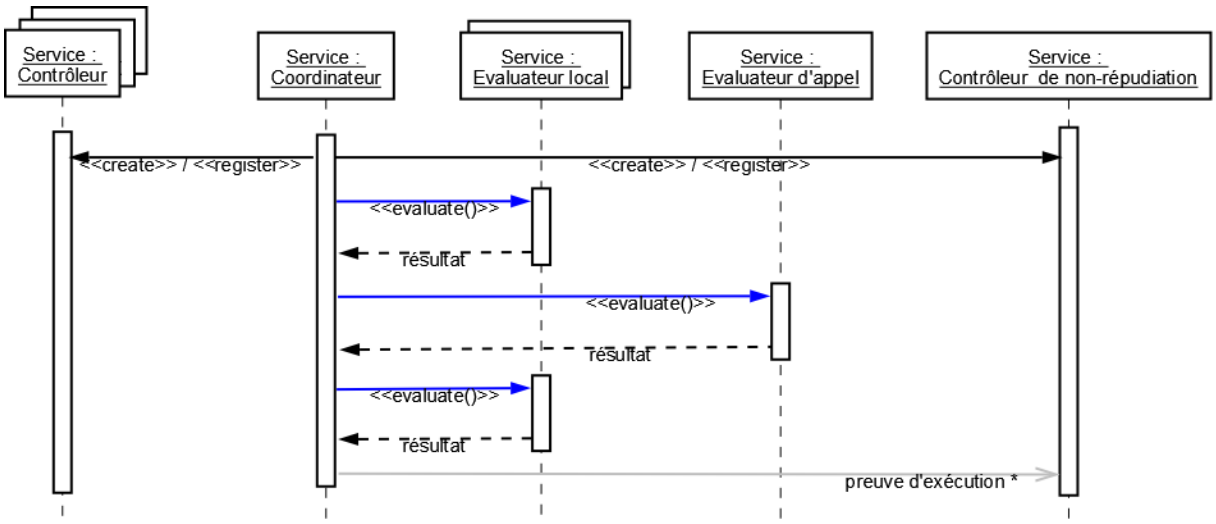


FIG. 5.7 – Interaction entre les composants de l'architecture d'exécution d'une activité

des évaluateurs globaux et d'autres contrôleurs qui ont trait à l'activité qu'il exécute. De même, il fournit des preuves de son exécution au(x) contrôleur(s) de non-répudiation. Tout cela permet d'assurer que son exécution respecte les contraintes globales. Le coordinateur utilise un évaluateur local pour l'évaluation des pré-conditions. En fonction du résultat rendu par cet évaluateur et des instructions reçues, il utilise un évaluateur d'appel pour l'évaluation de l'appel. En fonction du résultat rendu par l'évaluateur d'appel et des instructions reçues, il utilise un évaluateur local pour l'évaluation des post-conditions. En fonction du résultat d'évaluation des post-conditions, il va appeler le(s) coordinateur(s) qui exécute(nt) les activités successives. Le cas échéant, il appelle le contrôleur d'exception pour adapter son exécution.

5.2.2 Contrôleur d'interaction

Dans le cadre de l'exécution du plan, un composant appelle un autre composant en utilisant un contrôleur d'interaction. Ce dernier effectue des appels dont la structure est décrite dans la section 3.2.4. Dans MEOBI, l'interface Invocation, héritant de l'interface Information, définit les

manipulations sur cette structure. La fonction invoke définie par l'interface Service (introduite au début de ce chapitre) est la représentation la plus simple des contrôleurs d'interaction. Ces derniers sont donc automatiquement intégrés aux composants de l'architecture. Supposons qu'un service B veut appeler une fonction F fournie par un service A (F est différente de la fonction invoke). B appelle la fonction invoke de A en lui envoyant un message qui contient les informations caractérisant l'appel à F. La fonction invoke renvoie soit un message qui contient le résultat de l'exécution de F, soit un message qui contient une notification de l'état de cette exécution : avec succès ou défaillante. Cet appel requiert que A et B soient inter-opérables.

L'interface InteractionController de MEOBI est une représentation plus flexible des contrôleurs d'interaction. La figure 5.8 illustre les fonctions fournies par trois types de contrôleurs d'interaction : invocateurs, contrôleurs de partenaires et connecteurs. Ces fonctions sont utilisées pour l'exécution des appels de manière plus adaptable. En fonction des informations caractérisant l'appel à F, B peut créer ou s'abonner à un contrôleur d'interaction approprié pour l'exécution de l'appel. Cela permet de séparer et de bien gérer les phases de communication entre des services durant l'exécution d'un plan de coordination sécurisée d'activités. Cela permet également d'intégrer des mesures de sécurité pour cette communication.

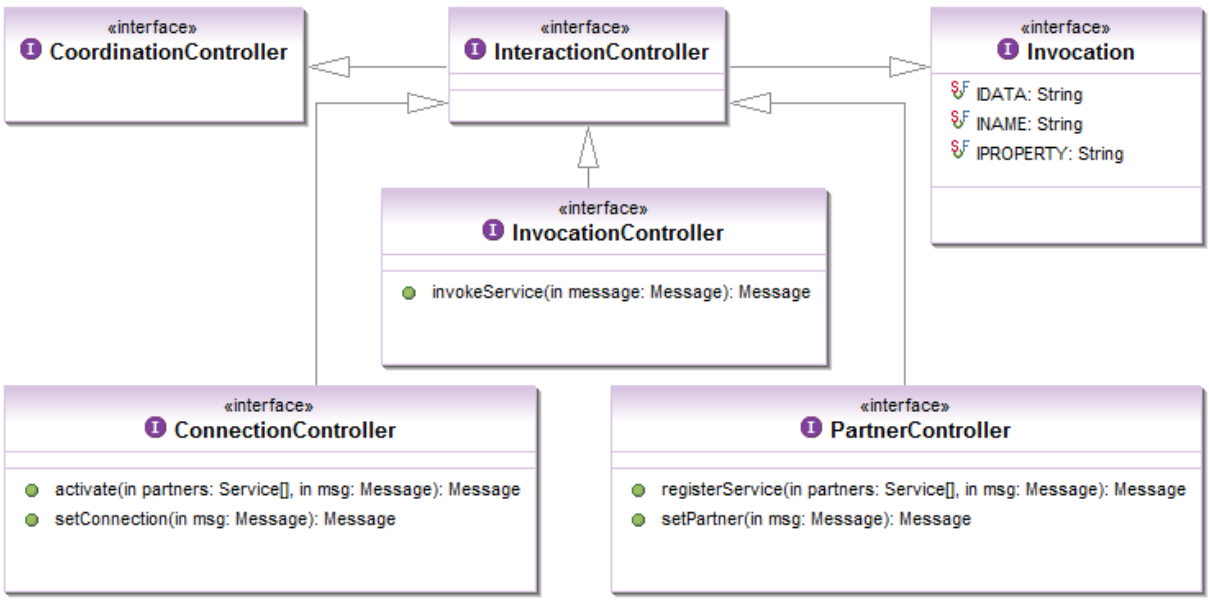


FIG. 5.8 – Diagramme de classes représentant les contrôleurs d'interaction

5.2.2a Invocateur

Un **invocateur** contrôle l'exécution de l'appel dans le cadre de l'activité. Dans la figure 5.8, l'interface InvocationController représente des invocateurs. La fonction invokeService permet d'appeler une fonction d'un service. Cet appel est conforme au mode d'emploi du service appelé. La figure 5.9 illustre l'architecture d'exécution de cet appel. L'invocateur réalise l'appel à l'aide d'un **connecteur** et d'un **contrôleur de partenaire**. Il renvoie le résultat d'appel à l'évaluateur d'appel de l'activité. Les connecteurs et les contrôleurs de partenaires utilisés par les invocateurs sont aussi des contrôleurs d'interaction.

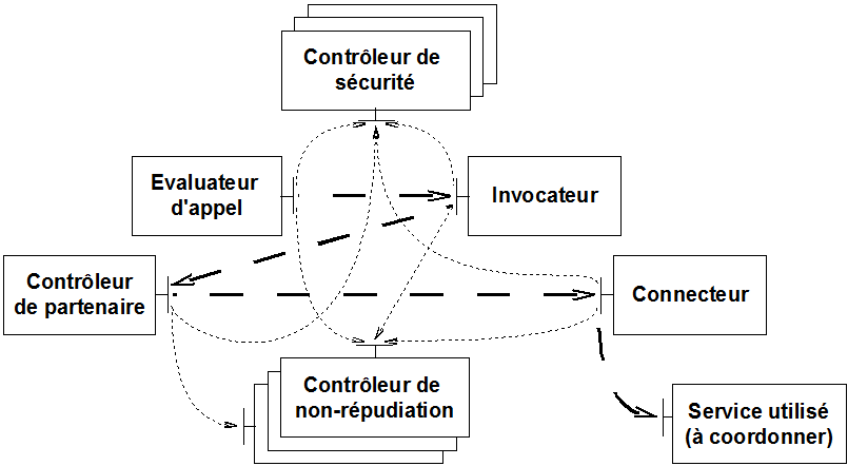


FIG. 5.9 – Architecture d'exécution d'un appel

5.2.2b Contrôleur de partenaires

Un contrôleur de partenaires est dédié à la gestion de services faisant partie de l'exécution d'un appel dans le cadre d'une activité. Comme illustrée dans la figure 5.8, la fonction registerService permet aux services de s'abonner auprès de l'invocateur qu'ils utilisent en se basant sur leurs modes d'emploi :

- les propriétés de déploiement de services, par exemple un service Web, un composant, etc.
- les propriétés et les contraintes représentant les rôles que les services jouent dans le moteur exécutant le plan.

La fonction setPartners permet d'enchaîner les services faisant partie de l'exécution de l'appel, en cas d'appel indirect (à travers des proxies, avec la présence de tiers de confiance, etc.).

5.2.2c Connecteur

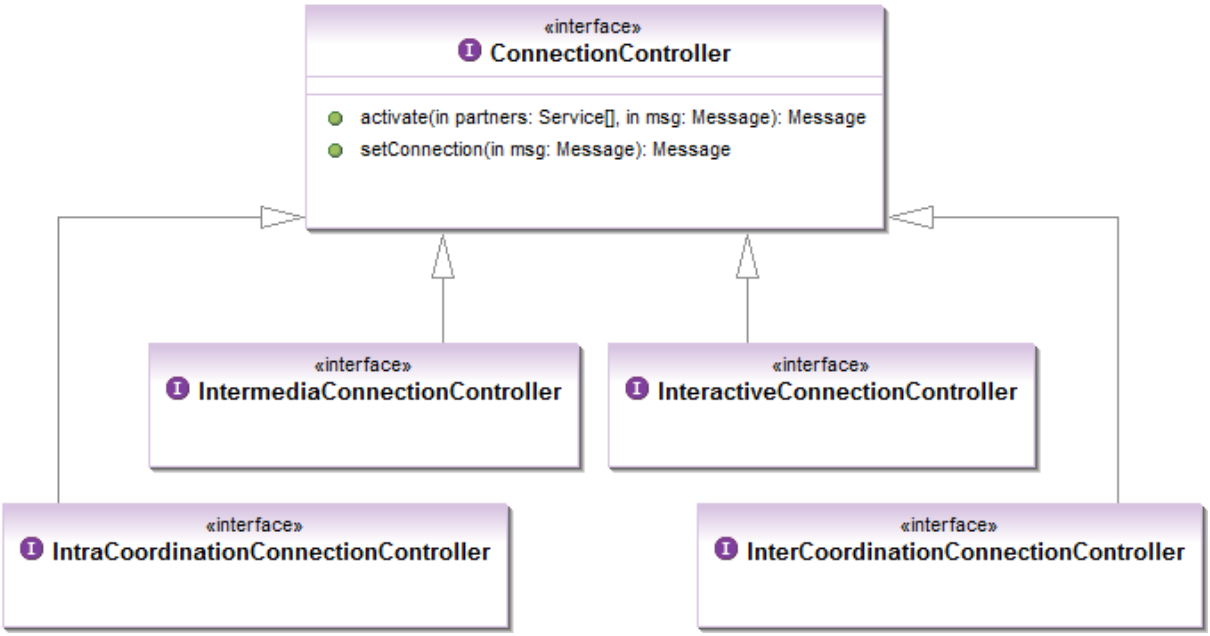


FIG. 5.10 – L'interface ConnectionController pour les connecteurs

Un connecteur sert à transmettre les données d'un appel ainsi que les données de résultat d'appel entre les services appelant et appelé. L'interface ConnectionController (voir la figure 5.10) représente des connecteurs. Quatre types de connecteurs correspondant aux quatre types de communication sont considérés : interactif (interface InteractiveConnectionController), via l'infrastructure de communication (interface IntermediaConnectionController), inter-coordination (interface InterCoordinationConnectionController) et intra-coordination (interface Intra-CoordinationConnectionController). Les connecteurs fournissent des fonctions qui portent sur :

- la gestion des différents types possibles de communication entre les services appelant et appelé, en se basant sur les propriétés et les contraintes de communication fournies par ces services, par exemple la fonction activate définie par l'interface ConnectionController.
- la création d'une session de communication convenable pour les services faisant partie de l'exécution d'une activité, par exemple la fonction setConnection définie par l'interface ConnectionController.

La figure 5.11 décrit des interactions principales entre un invocateur, un connecteur et un contrôleur de partenaires dans le cadre de l'exécution d'un appel.

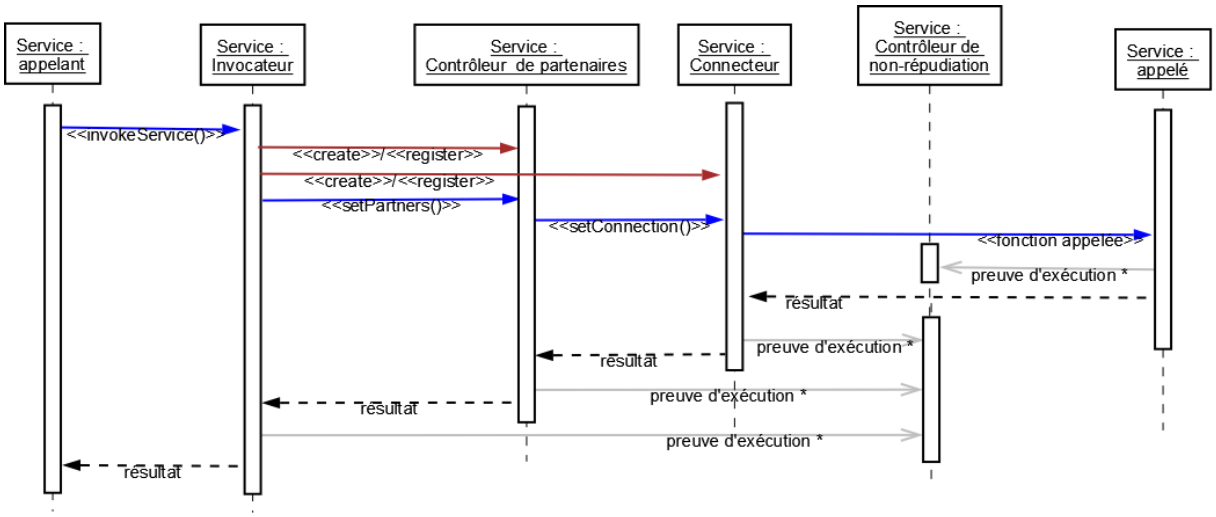


FIG. 5.11 – Interactions principales d'un invocateur

5.2.3 Évaluateur spécialisé

Un évaluateur spécialisé s'occupe de l'évaluation des contraintes spécialisées d'un aspect spécifique. La figure 5.12 illustre l'architecture d'exécution d'un évaluateur spécialisé. Ce dernier

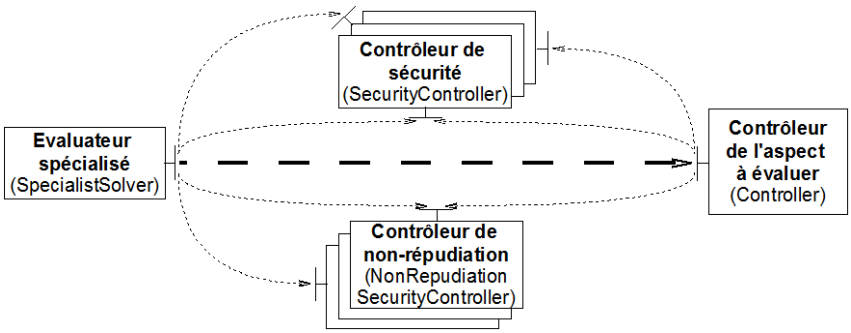


FIG. 5.12 – Architecture d'évaluation d'une contrainte spécialisée

prend en charge tout ou une partie du processus d'évaluation d'une contrainte spécialisée. Il peut

appeler d'autres services pour achever son évaluation. Pour cela, un évaluateur spécialisé crée ou s'abonne auprès des contrôleurs de cet aspect (conformément à la contrainte globale portant sur cet aspect). Il les utilise pour son évaluation, c.-à-d. pour construire la portée d'évaluation ou bien pour exécuter des opérateurs de contrainte.

Un évaluateur spécialisé fonctionne selon un des deux modes suivants : évaluation sur la portée statique ou évaluation sur la portée dynamique. Dans le premier cas, l'évaluateur construit par lui-même la portée d'évaluation de contrainte à partir des paramètres d'appel de l'évaluateur local. Dans le deuxième cas, l'évaluateur reconstruit la portée d'évaluation de contrainte en se basant sur les paramètres d'appel. La (re)construction est faite par l'évaluateur lui-même ou par un contrôleur de l'aspect correspondant ; cela requiert éventuellement l'interaction avec les contrôleurs de non-répudiation afin de récupérer les données nécessaires.

Après que la portée d'évaluation soit déterminée, un évaluateur spécialisé exécute les opérateurs de contrainte correspondant à la contrainte à évaluer. Il peut également faire exécuter ces opérateurs de contrainte aux contrôleurs de cet aspect.

Ensuite l'évaluateur spécialisé calcule le résultat de l'évaluation en fonction des résultats d'exécution des opérateurs de contrainte. Dans le cas d'une évaluation à *FAUX* , il notifie la non-satisfaction au contrôleur d'exception et attend ses instructions. Il produit des preuves d'exécution et les envoie à un contrôleur de non-répudiation spécialisé (contrôleur d'acquisition).

A l'exécution, un évaluateur spécialisé est uniquement appelé par un évaluateur local.

5.2.4 Évaluateur d'appel

Un évaluateur d'appel s'occupe de l'évaluation d'une famille d'appels. La figure 5.13 illustre les interactions principales d'un évaluateur d'appel. L'évaluateur d'appel utilise un invocateur pour construire la portée des contraintes à évaluer ; c'est le coordinateur correspondant qui s'est abonné à ou a créé cet invocateur. L'évaluateur d'appel fait appeler par cet invocateur la fonction de service à coordonner. Le résultat de l'appel est rendu à l'évaluateur d'appel. Ce

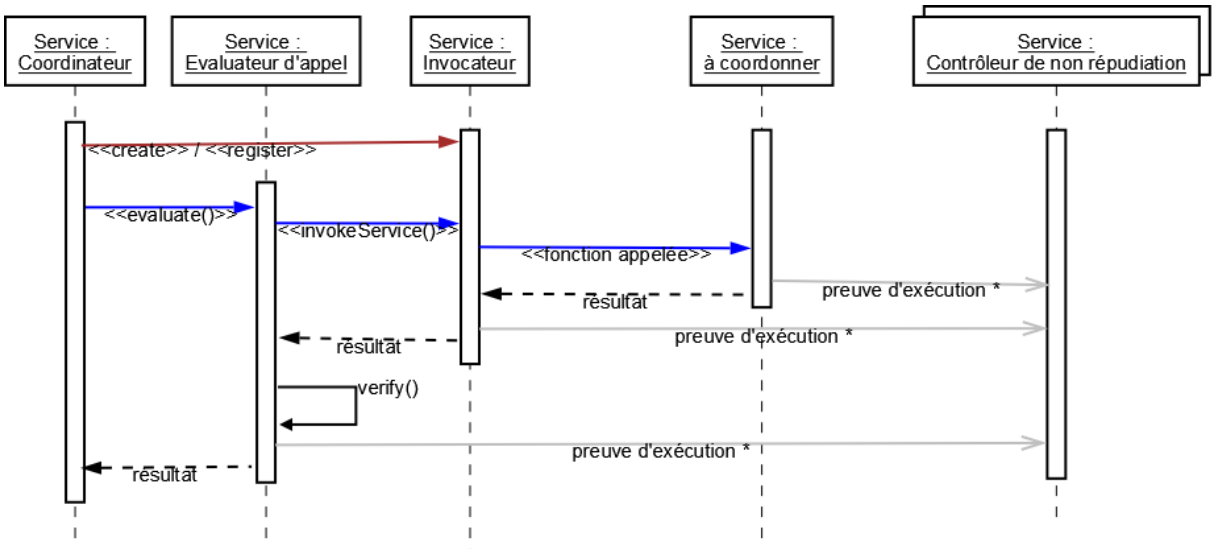


FIG. 5.13 – Interactions principales d'un évaluateur d'appel

dernier vérifie si ce résultat montre que l'exécution d'appel s'est bien passée. Puis il renvoie le résultat de vérification comme le résultat de l'évaluation d'appel. Avant de rendre les résultats, les évaluateurs, l'invocateur et éventuellement le service à coordonner notifient des preuves

d'exécution au contrôleur de non-répudiation auquel le coordinateur s'est abonné. Ils peuvent aussi s'inscrire auprès d'autres contrôleurs de non-répudiation. Dans ce cas, ils envoient des preuves d'exécution aux contrôleurs de non-répudiation auprès desquels ils se sont abonnés, ainsi qu'au contrôleur de non-répudiation auquel le coordinateur correspondant s'est abonné.

5.2.5 Évaluateur local

Il existe deux familles de contraintes locales : celles correspondant aux pré-conditions d'appel de l'activité et celles correspondant aux post-conditions d'appel de l'activité. Un évaluateur local s'occupe de l'évaluation d'une (ou des deux) famille(s) de contraintes locales. Comme illustré dans la figure 5.14, un évaluateur local est représenté par la classe abstraite LocalConstraintSolver. Cette dernière met en œvre la fonction evaluateByPosition (définie par la classe LocalConstraint) selon l'algorithme ALG. 10 décrit dans la section 4.5. Cela permet l'évaluation de pré-conditions ainsi que l'évaluation de post-conditions. La classe abstraite LocalConstraintSolver met en œvre également les fonctions register et handle (définie la classe Controller) ; cela permet à un évaluateur de recevoir des instructions venant d'autres composants de l'architecture durant son exécution.

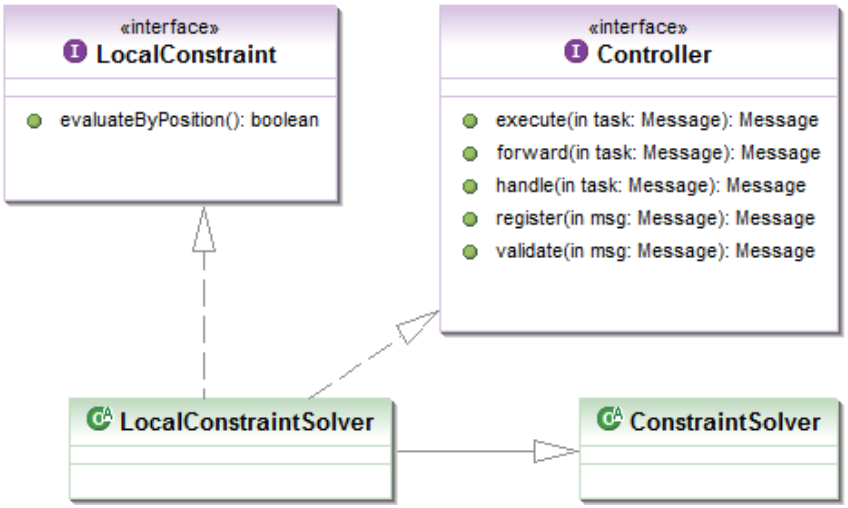


FIG. 5.14 – Diagramme de classes représentant les évaluateurs locaux

La figure 5.15 illustre les interactions principales d'un évaluateur local. Un évaluateur local est appelé par un coordinateur afin d'évaluer les pré-conditions ou les post-conditions d'appel de l'activité. L'évaluateur local utilise les évaluateurs spécialisés concernés pour obtenir les résultats d'évaluation de chaque contrainte spécialisée. Au cours de son exécution, il reçoit des notifications de la non-satisfaction venant des évaluateurs globaux concernés ; cela lui permet de lancer le traitement d'exception. Les interactions entre un évaluateur local et un évaluateur global sont détaillées dans la section 5.4.1.

5.3 Contrôleurs pour la gestion de la sécurité

Comme illustrée dans la figure 5.16, les contrôleurs de sécurité sont représentés par l'interface SecurityController. Les types de contrôleurs des aspects sécurité considérés du plan sont représentés par les interfaces qui héritent de SecurityController : contrôleurs d'authentification (resp. AuthenticationController), contrôleurs d'autorisation (resp. AuthorizationController), contrôleurs d'intégrité (resp. IntegrityController), contrôleurs de confidentialité (resp. ConfidentialityController) et

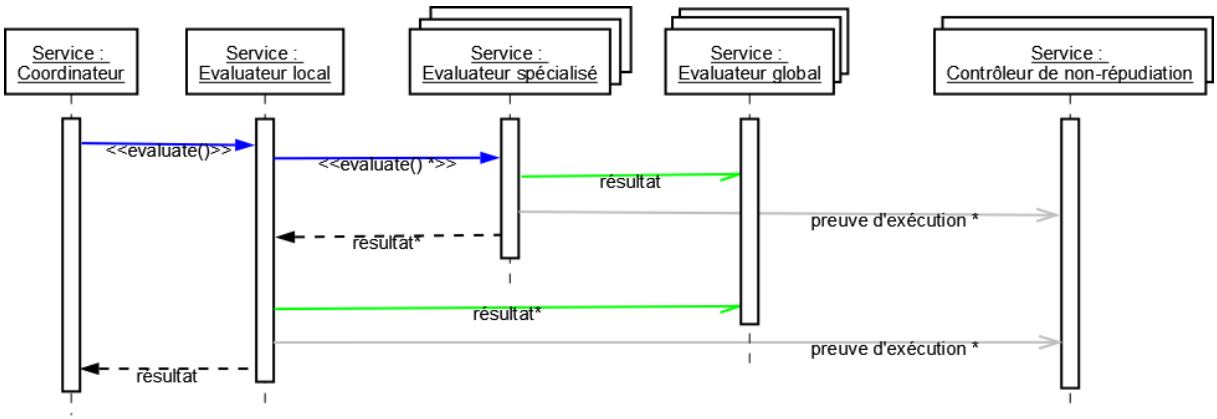


FIG. 5.15 – Interactions principales d'un évaluateur local

contrôleurs de non-répudiation (resp. NonRepudiationController). Un contrôleur de sécurité implante donc un mécanisme de sécurité qui est soit dédié à un aspect sécurité considéré du plan, soit générique à plusieurs aspects considérés du plan. Le cas échéant, il fait référence à un tiers de confiance qui fournit un mécanisme servant au même objectif.

Pour chaque aspect sécurité considéré du plan, nous distinguons les deux types de contrôleurs suivants en se basant sur leurs relations avec les évaluateurs :

- contrôleur servant à l'évaluateur spécialisé de cet aspect : la fonction execute d'un contrôleur de sécurité dédié à cet aspect est appelée par un évaluateur spécialisé pour réaliser son évaluation. Pour l'évaluation d'une contrainte spécialisée C (dans le cadre de l'exécution d'une activité A), un contrôleur de sécurité SC est configuré conformément aux paramètres de C et des propriétés (de l'aspect sécurité correspondant) de A. Cela est fait soit par l'évaluateur spécialisé (c.-à-d. l'évaluateur spécialisé crée le contrôleur qu'il utilise), soit par le biais du coordinateur associé (c.-à-d. l'évaluateur spécialisé utilise le contrôleur qui est créé par le coordinateur associé). La figure 5.17 illustre ces interactions.
- Contrôleur servant à l'évaluateur de la contrainte globale de cet aspect : un contrôleur de sécurité dédié à cet aspect assure l'exécution de composants de l'architecture conformément à la contrainte globale spécifiée. Cela est fait par le biais du coordinateur associé. Les fonctions validate et forward offrent cette possibilité. Les évaluateurs ainsi que les autres composants faisant partie de l'exécution du plan s'abonnent à ce contrôleur (fonction register) ; ils réalisent leurs activités sous le contrôle de ce contrôleur (fonction handle). La figure 5.18 illustre les interactions principales entre un contrôleur et un composant de l'architecture qui l'utilise. La figure 5.19 illustre les interactions entre un contrôleur de sécurité et un tiers de confiance à qui le contrôleur délègue ses activités.

Dans notre architecture, les contrôleurs des aspects sécurité ainsi que les contrôleurs d'autres aspects non-fonctionnels (tels que la sûreté de fonctionnement) ne sont pas le cœur de l'exécution d'un plan de coordination sécurisée d'activités. Ils sont présents pour aider (par eux mêmes ou par des tiers de confiance) les évaluateurs à achever leurs activités comme nous l'avons décrit ci-dessus. La mise en œuvre des fonctions définies par l'interface Controller permet aux contrôleurs de sécurité de réaliser leurs supports. De plus, la mise en œvre de la fonction invoke de l'interface Service permet aux composants de l'architecture d'utiliser (s'il y en a) les fonctions spécifiques des contrôleurs de sécurité. Les interactions entre ces contrôleurs et d'autres composants sont suivies des interactions décrites ci-dessus. Pour cette raison, dans la suite de cette section, nous abordons une grille de mesures de sécurité pouvant être instanciées sous formes de types de contrôleurs

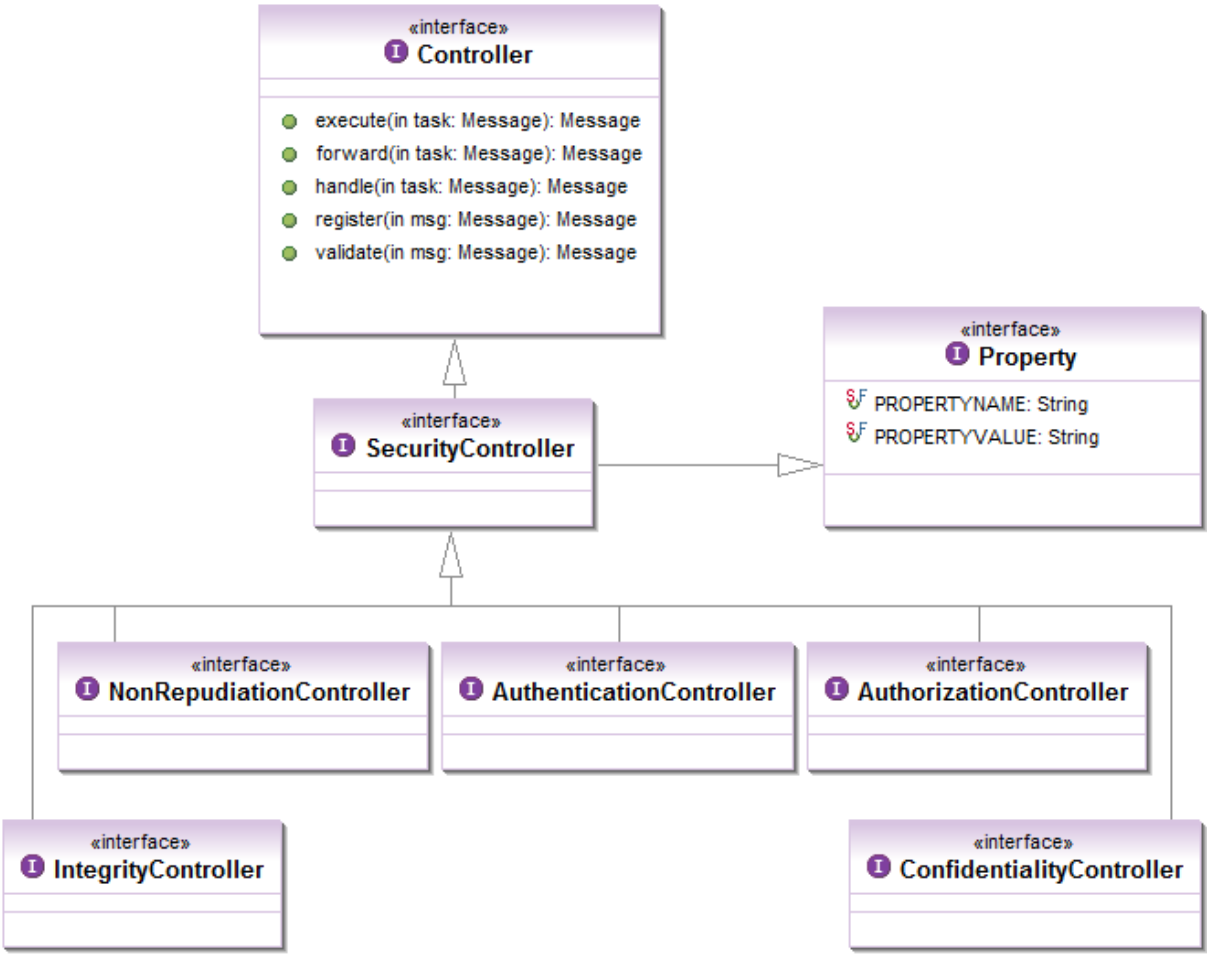


FIG. 5.16 – Diagramme de classes représentant les contrôleurs de sécurité

de sécurité plutôt que des fonctions définies par des interfaces qui héritent de Controller.

5.3.1 Contrôleur d'authentification

Les contrôleurs d'authentification sont représentés par l'interface AuthenticationController (voir la figure 5.20). Selon la caractérisation des mécanismes d'authentification présentée dans la section 2.2, nous distinguons deux types de contrôleurs d'authentification suivants :

- Les contrôleurs qui s'occupent de l'authentification des composants sont représentés par l'interface EntityAuthenticationController.
- Les contrôleurs qui s'occupent de l'authentification de la source des informations échangées entre les composants quand une session de communication est créée, sont représentés par l'interface InformationOriginController.

Un contrôleur d'authentification est activé conformément à la contrainte globale d'authentification d'un plan. Il est appelé par des évaluateurs pour l'évaluation des contraintes spécialisées d'authentification.

5.3.2 Contrôleur d'autorisation

La figure 5.21 illustre l'interface AuthorizationController qui représente les contrôleurs d'autorisation. Un contrôleur d'autorisation s'occupe de l'autorisation des appels, du contrôle d'accès aux informations définissant la coordination sécurisée. Il est utilisé par des évaluateurs de

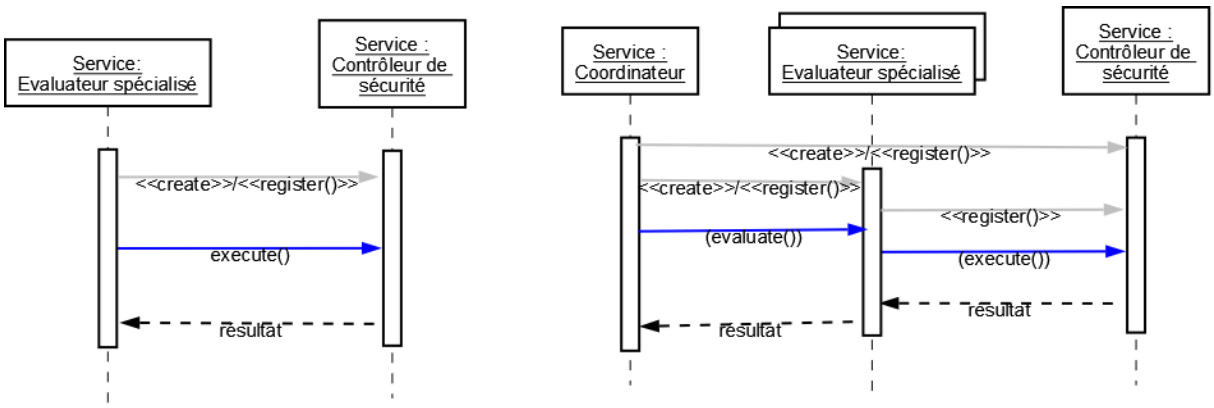


FIG. 5.17 – Interaction entre un évaluateur spécialisé de sécurité et un contrôleur de sécurité du même aspect

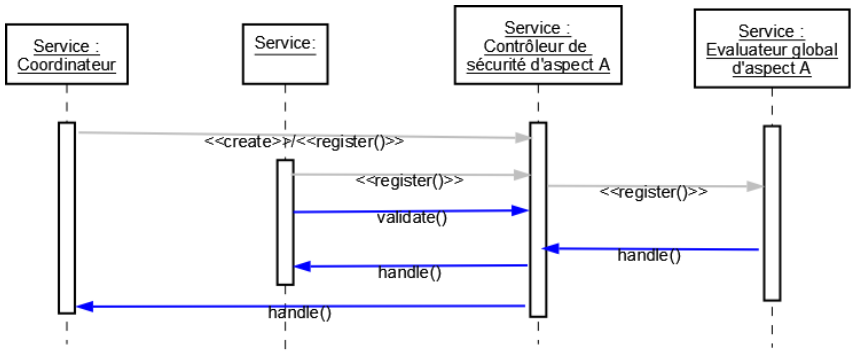


FIG. 5.18 – Interactions entre un évaluateur globale de sécurité et un contrôleur de sécurité du même aspect

contrainte pour valider la contrainte globale d'autorisation. Ses fonctionnalités portent sur deux axes principaux :

- Autorisation en fonction des mécanismes de sécurité : des mécanismes d'autorisation pouvant être instanciés sous forme des contrôleurs d'autorisation ont été présentés dans la section 2.3. Selon cette caractérisation, nous distinguons :
  - les contrôleurs d'autorisation qui implantent des mécanismes de contrôle d'accès (interface AccessController et les interfaces qui en hérite),
  - les contrôleurs d'autorisation qui implantent des mécanismes de gestion de confiance (interface TrustController).
- Autorisation en fonction de la satisfaction des contraintes spécifiées (interface SatisfactionDecisionController) : un contrôleur d'autorisation prend la décision d'autorisation en se basant sur le résultat de l'évaluation d'un évaluateur local.

5.3.3 Contrôleur d'intégrité

Les contrôleurs d'intégrité ont pour but d'assurer l'intégrité des informations échangées entre des composants de l'architecture (voir la figure 5.22).

L'interface ModificationBlocker représentent des contrôleurs d'intégrité mettant en œuvre les mécanismes qui empêchent des changements non autorisés (voir la section 2.4). Cela donne une possibilité de faire suivre les activités de ces contrôleurs d'intégrité aux contrôleurs d'autorisation qui implantent l'interface AccessController et vice-versa.

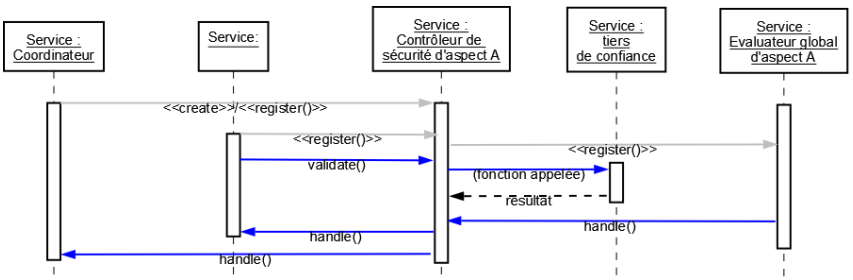


FIG. 5.19 – Interactions entre un contrôleur de sécurité et un tiers de confiance

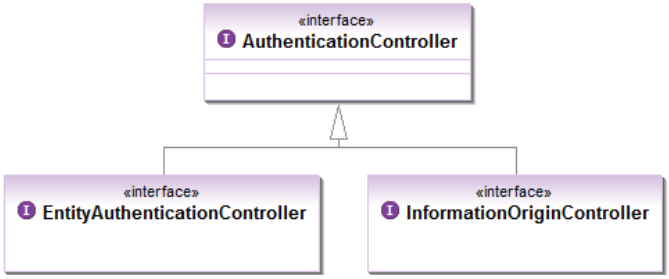


FIG. 5.20 – Diagramme de classes représentant les contrôleurs d'authentification

L'interface ModificationDetector représentent des contrôleurs d'intégrité mettant en œvre les mécanismes qui détectent des changements non autorisés. Cela donne une possibilité de faire suivre les activités de ces contrôleurs d'intégrité aux contrôleurs d'authentification qui implantent l'interface InformationOriginController et vice-versa.

Un contrôleur d'intégrité est utilisé par des évaluateurs spécialisés de contrainte d'intégrité pour valider une contrainte globale d'intégrité. En fonction de cette contrainte globale, il peut être utilisé par les contrôleurs d'interaction du modèle (invocateur, connecteur et contrôleur de partenaires) ; cela permet d'assurer les sessions de communication sécurisée entre les composants de l'architecture. Cela permet également d'établir les chaînes de confiances pour échanger les données entre ces composants.

5.3.4 Contrôleur de confidentialité

La figure 5.23 représente les contrôleurs de confidentialité. Ce contrôleur assure la confidentialité des données de l'application à base de services. Selon la caractérisation présentée dans la section 2.5, les types de contrôleurs d'intégrité suivants sont pris en compte :

- Les contrôleurs implantant des mécanismes de contrôle d'accès aux informations de l'exécution coordonnée des services : ils sont représentés par l'interface ConfidentialInformationAccessController qui hérite de ConfidentialityController. Un contrôleur de confidentialité de ce type peut éventuellement faire suivre ses activités par un contrôleur d'intégrité du type ModificationBlocker et vice-versa. De même, il peut éventuellement faire suivre ses activités par un contrôleur d'autorisation du type AccessController.
- Les contrôleurs implantant des mécanismes de chiffrement/déchiffrement de ces informations : ils sont représentés par l'interface EncryptionController.

5.3.5 Contrôleurs de non-répudiation

Comme nous avons déjà dit dans la section 2.6, la non-répudiation consiste à vérifier l'adéquation et la conformité du contrôle et de l'exécution des fonctions d'un service à ses contraintes

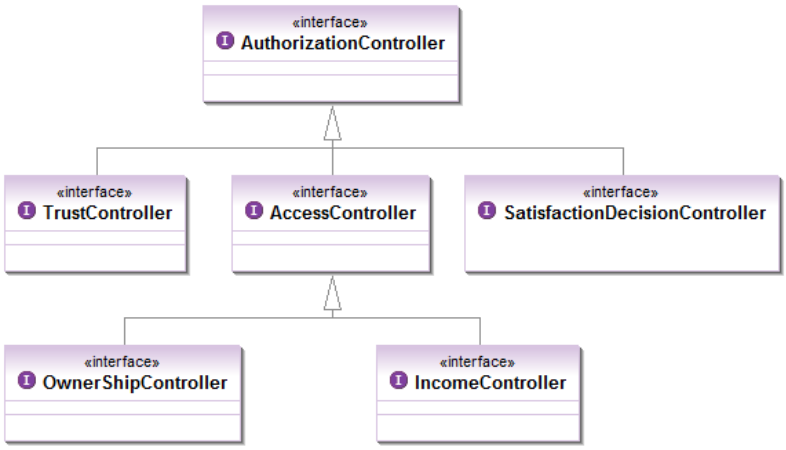


FIG. 5.21 – Diagramme de classes représentant les contrôleurs d'autorisation

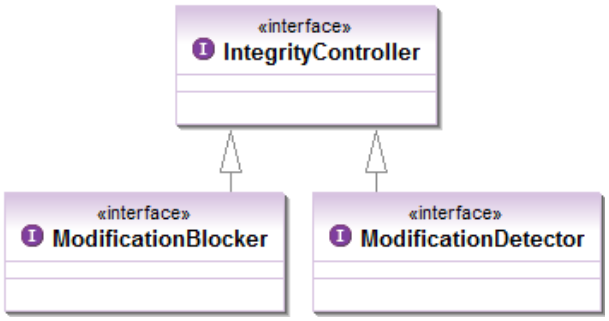


FIG. 5.22 – Diagramme de classes représentant les contrôleurs d'integrité

et ses propriétés. Dans le canevas MEOBI, la non-répudiation est contrôlée par des contrôleurs dédiés qui sont représentés par l'interface NonRepudiationController. Elle est également contrôlée par les évaluateurs globaux et les contrôleurs d'exception. La figure 5.24 présente deux types de contrôleurs de non-répudiation considérés : contrôleurs d'acquisition et contrôleurs de journalisation.

5.3.5a Contrôleur d'acquisition

L'interface Acquirer représente les contrôleurs d'acquisition. Un contrôleur d'acquisition permet à ses abonnés de collecter et/ou générer leurs preuves d'exécution (fonction startAcquisition) ; cela est fait en fonction des usages considérés de preuves d'exécution, en fonction des exceptions à détecter, en fonction de la répudiation à contrôler, etc. Les preuves sont collectées ou générées avec restriction au niveau de leur exploitation (fonction setProofWithRestriction) ou sans restriction au niveau de leur exploitation (fonction reportExecutionProof). Le contrôleur d'acquisition permet d'étiqueter les preuves collectées afin de faciliter leurs exploitations (fonction labelExecutionProof).

5.3.5b Contrôleur de journalisation

Un contrôleur de journalisation est représenté par l'interface Logger (voir la figure 5.24). Par le biais des contrôleurs d'acquisition décrits ci-dessus, les preuves d'exécution sont reportées par des composants de l'architecture durant l'exécution du plan. Les fonctions archiveProof, archiveProofWithRestriction servent à archiver ces preuves d'exécution. Il s'agit d'informations redondantes, mais elles sont très utiles pour le traitement des exceptions, pour l'exécution des activités successives du plan, et aussi pour des analyses ultérieures (par exemple l'évaluation

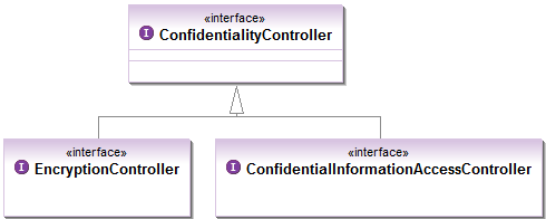


Fig. 5.23 – Diagramme de classes représentant les contrôleurs de confidentialité

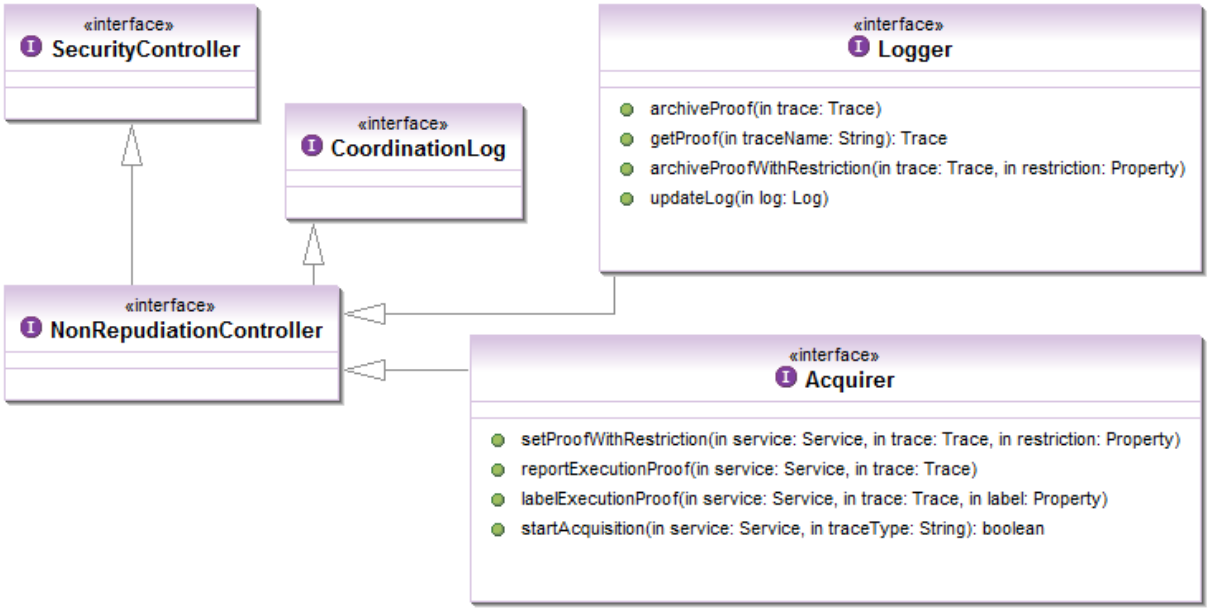


Fig. 5.24 – Diagramme de classes représentant les contrôleurs de non-répudiation

d'une contrainte globale). Ces informations concernent l'état d'exécution de chaque service, les opérations effectuées, les messages échangés, etc. Il archive et fournit ces preuves (fonction getProof) sur demande pour :

- le composant qui génère ces preuves ;
- le coordinateur de l'activité ;
- le contrôleur d'exception de l'activité ;
- les évaluateurs de contraintes spécialisées/globales des autres activités reliées ;
- les contrôleurs de sécurité et de sûreté de fonctionnement.

La figure 5.25 illustre les interactions principales entre un contrôleur de journalisation, un contrôleur d'acquisition et un composant de l'architecture. Ce dernier envoie à un contrôleur d'acquisition les preuves d'exécution avant de renvoyer le résultat d'exécution aux services qui l'appellent. Il peut attacher à ces preuves les informations qui restreignent les services qui peuvent récupérer ces preuves. Le contrôleur d'acquisition diffuse ces preuves d'exécution aux contrôleurs de journalisation auxquels il s'est abonné, ainsi qu'au contrôleur de journalisation auquel le coordinateur associé s'est abonné. Un composant appelle un contrôleur de journalisation pour récupérer des preuves qu'il a généré ou des preuves qu'il a le droit de voir. Par exemple, un évaluateur spécialisé appelle un contrôleur de journalisation pour construire une portée de la contrainte qu'il doit évaluer.

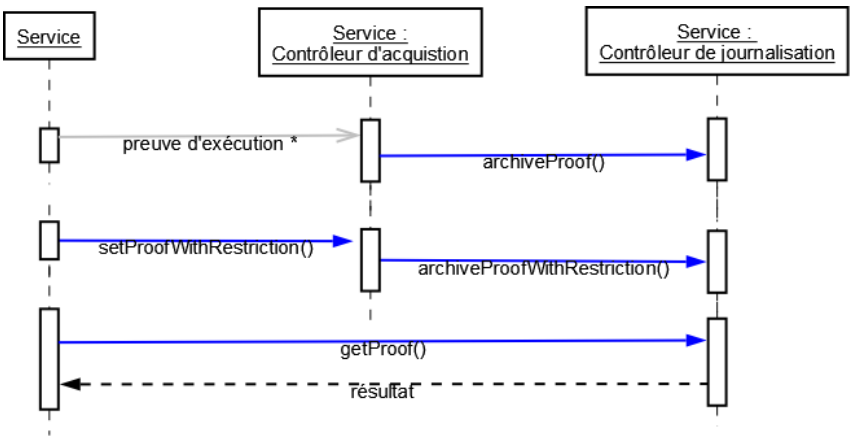


FIG. 5.25 – Interactions principales des contrôleurs de non-répudiation

5.4 Composants pour la sûreté de fonctionnement de la coordination sécurisée

La sûreté de fonctionnement de la coordination sécurisée est assurée à la fois par la satisfaction des contraintes et le traitement d'exceptions dans le cas de la non satisfaction d'une contrainte.

La sûreté de fonctionnement est systématiquement traitée au cours de l'évaluation des contraintes. Elle peut être adaptée en exprimant ses instructions en terme de contraintes d'adaptation spécifiques. Les contraintes globales et locales sont partiellement évaluées après l'évaluation fausse de chaque contrainte spécialisée concernée. Cela permet d'assurer l'inter-sûreté et l'intra-sûreté de l'exécution de chaque activité.

La validité des contraintes globales (de coordination, de sécurité, etc.) à l'exécution est ultérieurement vérifiée grâce aux informations extraites de journaux de coordination.

Les contrôleurs de sûreté de fonctionnement permettent d'assurer la sûreté de fonctionnement pour la gestion des composants de l'architecture. Ses fonctionnalités portent sur la surveillance des composants de l'architecture avec des notifications systématiques de changement au niveau de leurs propriétés et leurs contraintes (par exemple quand les propriétés d'un service sont ajoutées ou modifiées).

5.4.1 Évaluateur global

Un évaluateur global sert à l'évaluation d'une contrainte globale d'un aspect considéré du plan ; cette contrainte globale correspond aux contraintes spécialisées (de cet aspect) associées aux différentes activités. Cela implique l'affectation des valeurs effectives aux paramètres de contraintes globales par des évaluateurs spécialisés. L'interface GlobalConstraint (voir la figure 5.26) représente les évaluateurs globaux. Cette interface hérite de l'interface Constraint et de l'interface SafetyController. Cela permet à un évaluateur global de fonctionner à la fois en tant qu'évaluateur et en tant que contrôleur de sûreté de fonctionnement.

Concernant l'évaluation de la contrainte globale, un évaluateur global fournit des fonctions permettant l'évaluation totale (fonction evaluate) ou partielle (fonction partialEvaluate) d'une contrainte globale. Ces fonctions mettent en œvre des algorithmes respectivement présentés dans les sections 4.3 et 4.6. Il permet de réévaluer la contrainte globale pour contrôler la non-répudiation (fonction reEvaluate).

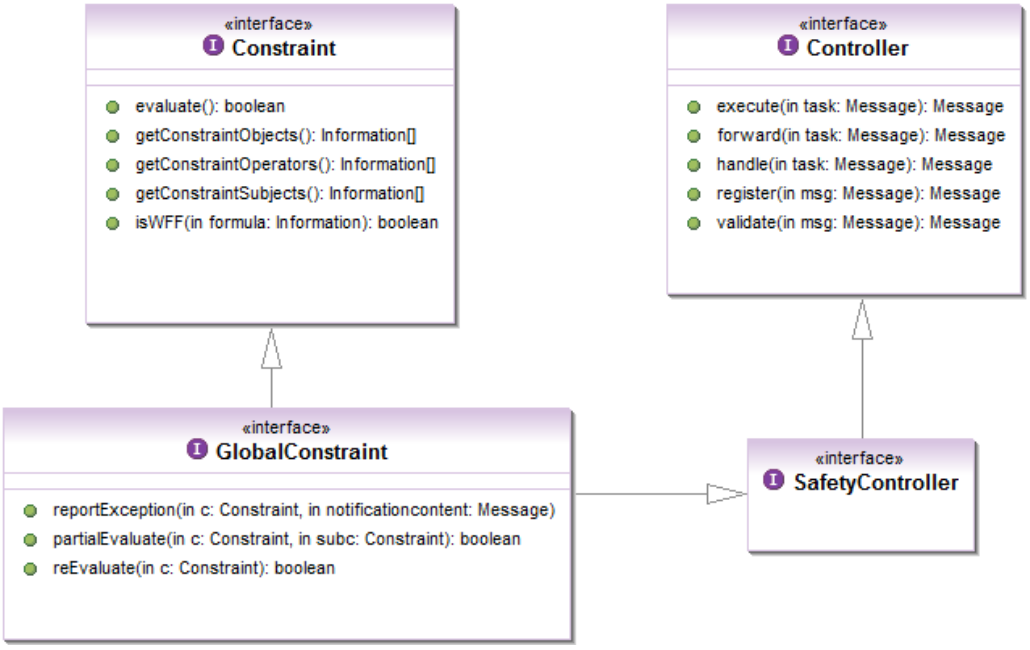


FIG. 5.26 – Diagramme de classes représentant les évaluateurs globaux

Les interactions entre un évaluateur global et les composants de l'architecture concernant le contrôle d'un aspect spécifique sont illustrées dans la figure 5.27.

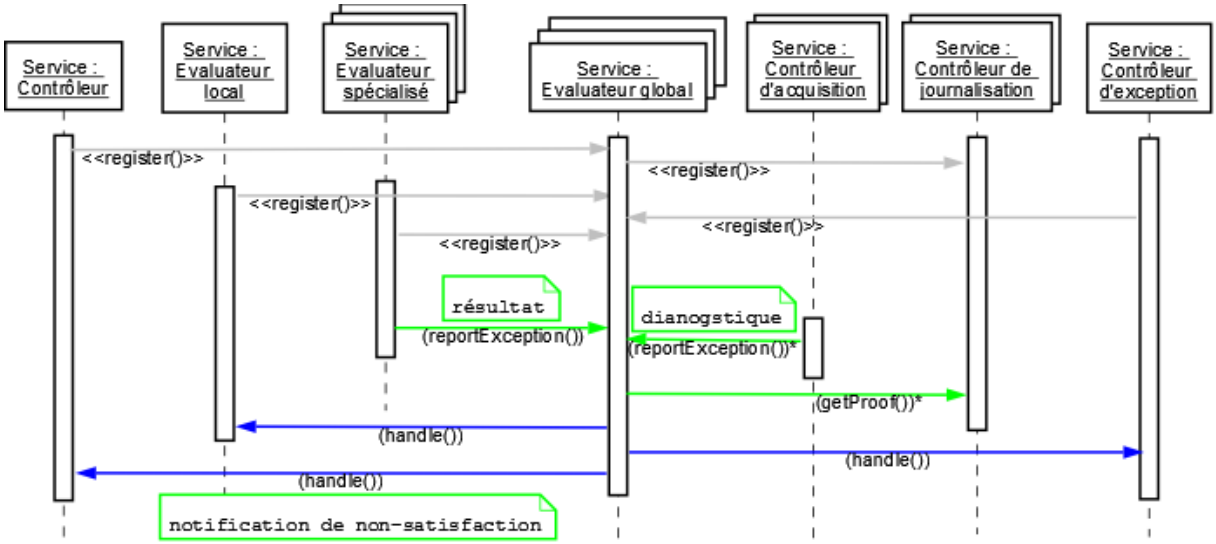


FIG. 5.27 – Interactions principales d'un évaluateur global

Tout d'abord, les composants de l'architecture s'abonnent auprès de l'évaluateur global en appelant la fonction register. Durant l'exécution d'une activité, les évaluateurs spécialisés abonnés envoient la notification de la non satisfaction d'une de ses contraintes spécialisées constituantes (fonction reportException). Chaque fois que l'évaluateur global reçoit cette notification, il évalue partiellement la contrainte globale dont il s'occupe ; cela a pour but de vérifier si cette évaluation fausse implique ou non l'évaluation à *FAUX* de la contrainte globale. De même, en fonction des diagnostics sur cet aspect qui sont envoyés par les contrôleurs d'acquisition (fonction reportException), un évaluateur global récupère des preuves d'exécution à partir de contrôleurs de journalisation.

Ensuite il élimine des informations reçues redondantes et filtre les informations incohérentes.

Toutes ces informations sont comparées, analysées (en fonction de la contrainte globale correspondante) pour détecter si une exception a probablement eu lieu. En fonction de l'évaluation partielle ainsi que l'analyse des diagnostics et des preuves d'exécution, le contrôleur global notifie des exceptions en temps réel aux contrôleurs abonnés et aux contrôleurs d'exception auxquels il s'abonne ; cela se fait en appelant la fonction handle de ces contrôleurs. Cet évaluateur global permet également de notifier en temps réel (par la paramétrisation de la même fonction) la non-nécessité de réévaluer une contrainte spécialisée déjà satisfaite aux évaluateurs locaux abonnés.

5.4.2 Contrôleur d'exception

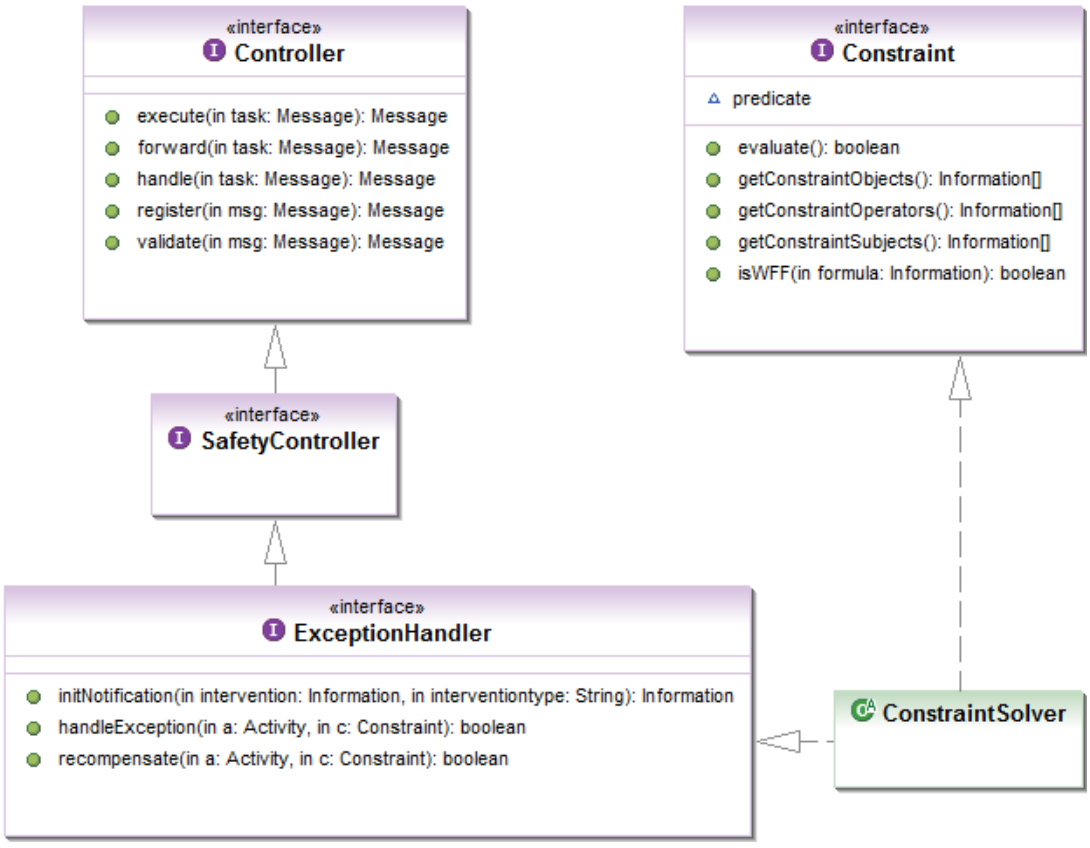


FIG. 5.28 – Diagramme de classes représentant les contrôleurs d'exception

Dans la figure 5.28, les contrôleurs d'exception sont représentés par l'interface ExceptionHandler.

Un contrôleur d'exception met en œuvre le traitement des exceptions (du même type) survenues lors de l'évaluation d'une contrainte. Un contrôleur d'exception est donc spécifique à un des types d'exceptions suivants :

- La non-satisfaction d'une contrainte spécialisée ;
- La non-satisfaction d'une contrainte spécialisée qui provoque la non-satisfaction d'une contrainte locale ;
- La non-satisfaction d'une contrainte spécialisée qui provoque la non-satisfaction d'une contrainte globale ;
- La non-satisfaction d'une contrainte d'appel.

Les interactions principales entre un contrôleur d'exception et des composants de l'architecture sont illustrées dans la figure 5.29. Un contrôleur d'exception reçoit la notification de la présence

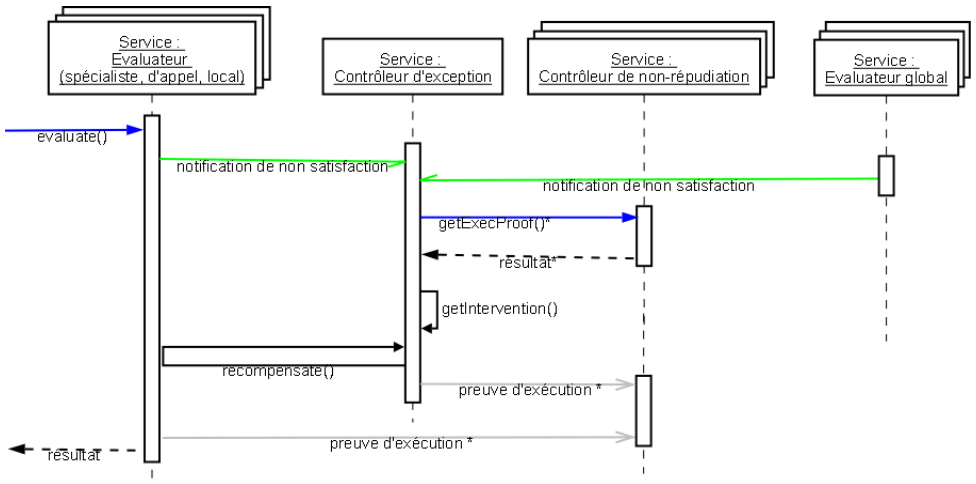


FIG. 5.29 – Interactions principales d'un contrôleur d'exception

de l'exception et déclenche un processus de traitement d'exception. Ce processus est spécifique au contrôleur d'exception concerné ; il utilise des preuves d'exécution archivées par le contrôleur de journalisation, ainsi que les interventions interceptées (interventions internes, par exemple à partir de l'évaluateur global ou coordinateur, interventions externes, par exemple à partir de contraintes redéfinies de l'utilisateur). Le contrôleur d'exception collecte des preuves de l'état d'exécution avec succès à partir des journaux de coordination (gérés par le contrôleur de non-répudiation). Cela est fait lors de la réception d'une notification de la non-satisfaction à partir d'un évaluateur inscrit (que ce soit un évaluateur spécialisé, local ou d'appel), et/ou la notification de la non-satisfaction d'un évaluateur global correspondant. Ensuite il est en attente des interventions qui renseignent des informations utiles pour régir l'exécution par la suite : annuler, continuer ou reprendre l'évaluation.

5.4.3 Contrôleur d'adaptation

Pour assurer la sûreté de fonctionnement de l'exécution du plan, il est souhaitable d'autoriser et gérer la modification des composants faisant partie de cette exécution. L'adaptation d'un composant sur un aspect spécifique, qu'il soit fonctionnel (par exemple adaptation des propriétés de communication pour l'inter-opérabilité entre les services) ou non-fonctionnel (par exemple adaptation des propriétés d'authentification) correspond à un des cas suivants :

- la modification des propriétés d'un aspect de service en respectant ses contraintes internes spécifiées ;
- la modification des contraintes d'un aspect de service en retenant ses propriétés spécifiées ;
- la modification des contraintes et des propriétés d'un aspect de service ;
- l'ajout des contraintes, des propriétés et des fonctions d'un nouvel aspect en respectant le mode d'emploi spécifié.

L'interface AdaptationController illustrée dans la figure 5.30 représente des contrôleurs d'adaptation permettant de réaliser ces modifications. Les deux types de contrôleurs d'adaptation suivants sont considérés : générateurs d'informations et gestionnaires des aspects.

5.4.3a Générateur

L'interface InformationGenerator représente les générateurs d'informations. Les fonctions generate et restructure de cette interface permettent à un générateur de générer et (re)structurer une in-

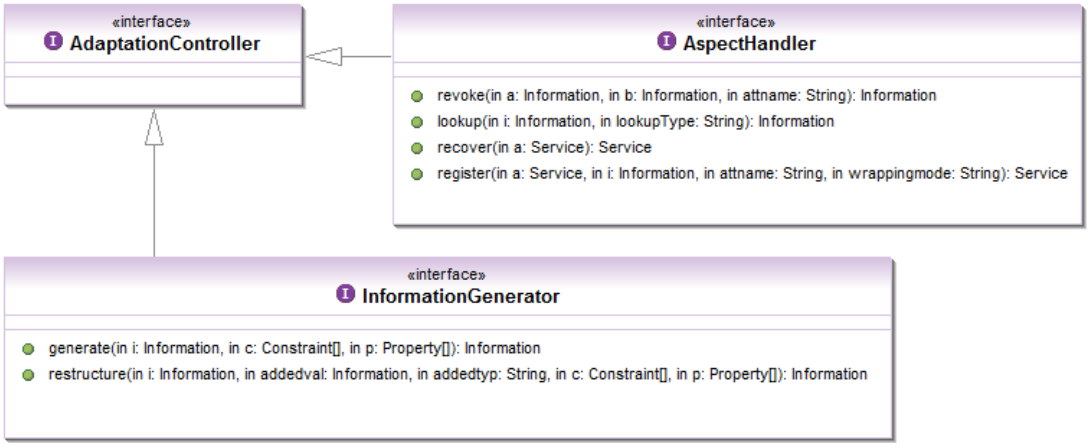


FIG. 5.30 – Diagramme de classes représentant les contrôleurs d'adaptation

formation. Les informations produites par ces fonctions sont classifiées en deux groupes :

- Celles qui visent à examiner la capacité (de communication, de sécurité, etc.) d'un service avant la coordination : ce sont des conditions nécessaires que le service doit satisfaire pour participer à l'exécution coordonnée conformément au plan. Ces informations sont produites conformément à une(des) contrainte(s) d'un plan.
- Celles qui visent à assurer un critère de qualité (de communication, de sécurité, etc.) d'un service et sa validation quand ses fonctions sont appelées dans le cadre de l'exécution des activités : ce sont des conditions suffisantes de services quand ses fonctions sont exécutées et/ou le résultat de l'exécution attendu par le service appelant. Ces informations sont produites conformément aux contraintes internes d'un service utilisé.

5.4.3b Contrôleur d'aspect

Ce contrôleur a pour but de gérer les propriétés d'un aspect qui sont dynamiquement associées à un service. Autrement dit, il gère les affectations potentielles aux paramètres de contraintes. La fonction register permet d'attacher une propriété à un service. De cette façon, elle permet à un service de créer ou de s'abonner automatiquement aux contrôleurs décrits par une contrainte globale de l'aspect correspondant. L'attachement est conforme à un des deux modes suivants :

- Attachement temporaire : cette information est valable seulement dans le cadre de l'exécution d'une activité. Elle est utilisée comme l'affectation de valeurs aux paramètres d'une contrainte spécialisée devant être évaluée dans le cadre de l'activité concernée ;
- Attachement permanent : cette information est valable dans le cadre de l'exécution du plan. Elle est utilisée comme l'affectation de valeurs aux paramètres d'une contrainte spécialisée à évaluer dans le cadre de l'activité concernée, ainsi que pour l'évaluation ou la réévaluation totale de la contrainte globale correspondante.

La fonction lookup permet de récupérer les informations sur une propriété qui a été attachée. La fonction revoke permet de détacher une propriété qui a été attachée. De cette façon, elle permet à un service de se désinscrire automatiquement des contrôleurs décrits par une contrainte globale de l'aspect correspondant. La fonction recover permet de reprendre toute propriété initiale du service.

5.5 Instanciation du canevas

5.5.1 Règles d'instanciation

Un plan de coordination sécurisée d'activité est exécuté par une configuration spécifique de notre architecture. Dans cette configuration, les composants sont instanciés selon les règles suivantes :

- **Service :**
  - Chaque service constitutif de la configuration est associé à au moins une instance de connecteur. Cela permet à un service de répondre aux appels d'autres services conformément aux différents protocoles de communication. Une instance de connecteur est requise pour des appels dont les propriétés et les contraintes de communication sont identiques.
  - Un service peut utiliser une ou plusieurs instances du contrôleur d'acquisition pour collecter et/ou générer ses preuves d'exécution. Une instance de contrôleur d'acquisition doit être présente pour chaque type d'usage des preuves d'exécution servant à l'évaluation de la contrainte globale.
  - Un service peut être associé à une instance du service adaptateur afin d'homogénéiser les propriétés fournies et requises dans le cadre de l'exécution du plan.
- **Activité :** Une activité du plan de coordination sécurisée est exécutée par une instance du service coordinateur. Cette exécution requiert également la présence :
  - d'au moins une instance de l'évaluateur local (pour l'évaluation de pré-conditions et/ou de post-conditions) ;
  - d'une instance de l'évaluateur d'appel ;
  - d'au moins une instance de contrôleur de journalisation.
- **Appel :** L'exécution d'un appel dans le cadre d'une activité correspondante requiert la présence :
  - d'une instance d'invocateur ;
  - d'une instance de contrôleur de partenaire ;
  - d'une instance de connecteur.
- Chaque aspect de l'exécution du plan (ou d'une activité) requiert au moins :
  - une instance de l'évaluateur spécialisé ;
  - une instance de l'évaluateur global ;
  - une instance de contrôleur de cet aspect.

5.5.2 Implantation

Cette section présente l'implantation que nous avons conduite pour valider notre canevas MEOBI. L'idée de cette implantation est de bien montrer la contribution principale de notre modèle MEO, c.-à-d. la capacité de décrire d'une manière uniforme des aspects fonctionnels et non-fonctionnels d'un plan de coordination. Ainsi, il faut montrer l'exécution d'un plan de coordination en fonction des évaluateurs des contraintes. Pour atteindre ces objectifs, nous offrons une implantation partielle de MEOBI en Java [jav], en utilisant l'environnement Eclipse [ecl]. Cette implantation se compose de 24 366 lignes de code. Elle comprend :

- La réécriture des interfaces MEOBI (qui sont spécifiées de manière indépendante aux langages de programmation) en 52 classes interfaces Java.
- L'implantation des types d'information spécifiant le plan en 21 classes Java concrètes.

– L'implantation des composants de l'architecture proposée en 27 classes Java abstraites et 47 classes Java concrètes.

Cette implantation nous permet d'instancier des moteurs qui sont capables d'exécuter des plans de coordination non sécurisée et sécurisée de services qui sont supposés inter-opérables. Les composants de chaque moteur respectent des règles de communication entre des composants de l'architecture proposée. La mise en œuvre de ces moteurs ne vise pas à résoudre le problème d'inter-opérabilité entre des services utilisés ; des propriétés et des contraintes d'exécution et de déploiement permettant l'inter-opérabilité de ces services doivent être décrites a priori. Les sections suivantes détaillent la mise en œuvre de certains composants de cette architecture.

Mise en œuvre des évaluateurs Les évaluateurs suivants ont été implantés sous forme les surcharges de la fonction evaluate() de l'interface Constraint :

– Évaluateurs spécialisés : chaque évaluateur spécialisé implanté permet d'évaluer un prédicat spécifique qui est défini dans le chapitre 3. L'évaluation de tout prédicat prédéfini a été implantée. Ces évaluateurs spécialisés sont compatibles en cas d'évaluation du prédicat sur une portée spécifique, qu'elle soit statique ou dynamique. Par ailleurs, nous implantons les évaluateurs spécialisés en fournissant les fonctions correspondantes aux connecteurs et quantificateurs logiques.
– Évaluateurs locaux : un évaluateur local permettant d'évaluer la conjonction des contraintes spécialisées de même aspect a été implanté. Il peut être instancié pour tout aspect considéré.
– Évaluateurs globaux : un évaluateur global fournissant les fonctions correspondant aux connecteurs et quantificateurs logiques a été implanté. Il est compatible en cas d'évaluation partielle et d'évaluation totale d'une contrainte globale. Il peut être instancié pour tout aspect considéré.

Mise en œuvre des contrôleurs

– Connecteur : un connecteur a été implanté. Il peut être instancié pour toute activité à exécuter.
– Invocateur : un invocateur simple a été implanté sous la forme de la fonction invoke() de l'interface Service. Il est à instancier pour tout composant utilisé.
– Gestionnaire des partenaires : les fonctions permettant d'ajouter (s'inscrire) ou supprimer les services faisant partie de l'exécution d'une activité de ce service ont été implantées.
– Contrôleur : les contrôleurs de sécurité ont été implantés. Nous simulons les mécanismes de sécurité comme présenté dans le chapitre 2. Chaque mécanisme est implanté par un contrôleur de sécurité spécifique. Il va être indépendamment instancié par les évaluateurs de sécurité et pour surveiller l'exécution des composants.
– Adaptateur : les fonctions de gestion des propriétés d'un service sont implantées.

5.5.3 Validation expérimentale

Nous avons démontré une application à base de services qui est construite à l'aide de MEOBI. L'objectif principal est de vérifier la validité de l'exécution du plan (en temps réel ou à postériori) en fonction de :

– Détection de la falsification des services dans le cadre de l'exécution des activités d'un plan ;

– Détection de l'interception des messages échangés dans le cadre de l'exécution des activités d'un plan ;
– Détection de l'exécution des activités en désordre.

La validation expérimentale consiste en la mise en œuvre d'une coordination non-sécurisée et d'une coordination sécurisée. Cela a pour but de développer une application de réservation de vols en utilisant l'implantation de notre canevas MEOBI.

Les services utilisés pour construire une telle application sont décrits dans la section 3.4.2. La logique applicative et la spécification du plan ont été également décrites dans cette section.

Pour valider notre proposition, nous utilisons un service FakeBank qui fournit une même interface et mode d'emploi que le service Bank. La différence est qu'il ne vérifie pas le solde du compte de client, il donne toujours l'autorisation de paiement. Ensuite nous présentons la description du plan de coordination. Nous montrons des vulnérabilités qui influencent la sûreté de fonctionnement du moteur d'exécution de ce plan. Nous montrons pourquoi des mesures de sécurité actuelles ne peuvent pas couvrir ces vulnérabilités.

Nous présentons ensuite les configurations de l'architecture correspondant à un plan de coordination non sécurisée et un plan de coordination sécurisée.

Nous montrons la sûreté de fonctionnement du moteur d'exécution de ce plan. En fin, nous discussions les avantages que MEOBI apporte au développeur d'une telle application et concluons la section.

Descriptions de services utilisés Les services suivants sont utilisés :

• **Agence :** Le service *Agence* fournit des fonctions (voir Fig. 3.10) permettant l'interaction et la gestion des clients qui s'intéressent à la réservation des places, la recherche des places disponibles ainsi que la pré-réservation des places choisies sur des bases de données de vols de différents opérateurs. Le mode d'emploi de ce service spécifie les aspects fonctionnels et non-fonctionnels de son exécution : sa preuve de contrôle d'accès, la visibilité de ses fonctions, leur ordre d'appel à respecter ainsi que le service d'interaction utilisé lors de la communication, etc. La figure 3.11 représente une partie de son mode d'emploi.

• **Bank, Fake Bank :** Le service *Bank* fournit des fonctions permettant de vérifier le solde d'un compte bancaire d'un client, de débiter le compte bancaire d'un montant ou de créditer un montant à un compte bancaire d'un client. Ces fonctions intègrent l'authentification du titulaire de ce compte.

Le service *Fake Bank* est un service qui fournit des fonctions similaires à *Bank*. La différence est que ce service ne fait pas la vérification du solde d'un compte bancaire d'un client ; il renvoie toujours l'autorisation de paiement, quel que soit le solde du compte bancaire du client.

signatures : set(authenticate : (*id* : authenSubjProof) × (*credential* : authenObjProof) → (*authorisation* : Boolean))

Fig. 5.31 – Fonctions à coordonner, fournies par le service AuthenServer

• **Authentication Server :** Le service *Authentication Server*, héritant de l'interface AuthenticationController, fournit des fonctions permettant d'authentifier un service avec un type de preuve prédéfini.

• **Application de réservation de vols :** L'application de réservation de vols est construite à partir de deux services existants *Agence* et *Bank*. En appelant des fonctions de ces services, l'application à base de services effectue les fonctions suivantes :

- chercher une liste des places disponibles selon les besoins de clients ;
- réserver une place en choisissant à partir de la liste des places disponibles ;
- acheter un billet.

Logique d'application Une logique d'application simple est décrite dans la section 3.4.2b.

Exécution d'un plan de coordination non sécurisée d'activités Cette spécification est transformée en un plan de coordination exécutable (voir la section 4.2) qui conduit une configuration dynamique des composants de l'architecture d'exécution. La figure 5.32 représente une configuration de l'architecture proposée pour l'exécution d'un plan de coordination non sécurisée. Il s'agit des groupes des évaluateurs de contraintes, des invocateurs et des contrôleurs

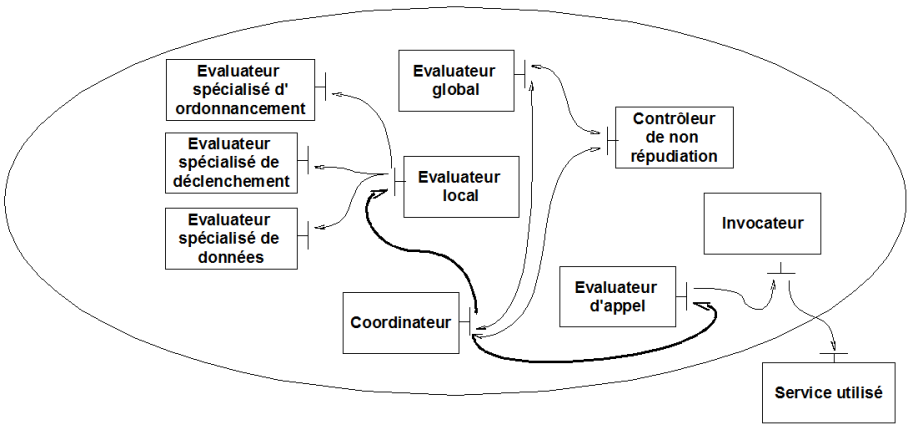


FIG. 5.32 – Instance du canevas pour l'exécution d'un plan de coordination non sécurisée

de coordination et de non-répudiation servant à l'exécution de chaque activité. La communication entre ces composants respecte les règles d'interaction décrites dans les sections 5.2 à 5.4.

Vulnérabilités démontrées Les vulnérabilités suivantes sont démontrées :

- L'appel à la fonction verifyAccountBalance du service Bank est redirigé vers le service *Fake Bank*. Ce service renvoie toujours l'autorisation de paiement, quel que soit le solde du compte bancaire du client.
- Le contenu d'un message échangé est modifié : le montant du débit est baissé.
- Le résultat d'exécution d'une activité est répudié : un dossier de préréservation devient un dossier de réservation payé.

L'exécution du plan de coordination se passe normalement parce que les contraintes de sécurité ne sont pas imposées.

Exécution d'un plan de coordination sécurisée d'activités La figure 5.33 représente une configuration de l'architecture proposée pour l'exécution d'un plan de coordination sécurisée. Dans cette configuration, les évaluateurs et contrôleurs de sécurité servant à l'évaluation des contraintes de sécurité sont ajoutés. Le processus d'exécution reste inchangé. La communication entre ces composants respecte les règles d'interaction décrites dans les sections 5.2 à 5.4. Des aspects de coordination et de sécurité du plan de coordination sécurisée sont spécifiés sous

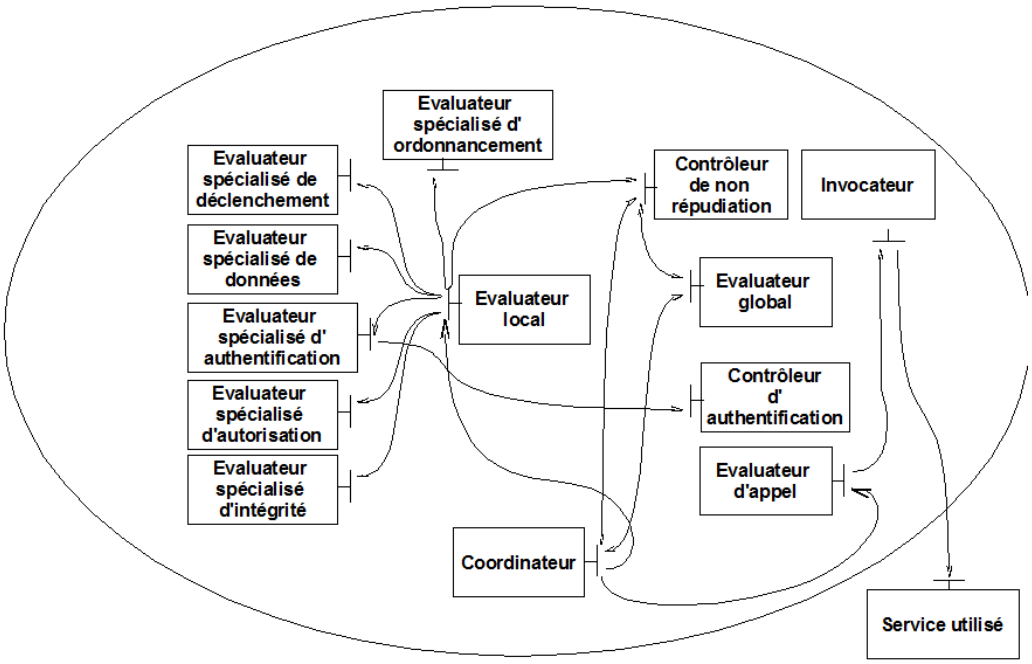


Fig. 5.33 – Instance du canevas pour l'exécution d'un plan de coordination sécurisée

forme de contraintes sur des activités à coordonner. Ces contraintes sont accompagnées par des informations servant à la génération du plan et à leur évaluation. Lors de l'exécution, les vulnérabilités démontrées ci-dessous ne peuvent pas se passer.

Puisque l'évaluation de contrainte d'authentification notifie que le service FakeBank n'est pas le service prévu à appeler, l'exécution de l'activité correspondante est annulée.

Puisque l'évaluation de contrainte d'intégrité notifie que le contenu de message échangé est modifié, l'exécution de l'activité correspondante est annulée.

Puisque l'évaluation de contrainte de non-répudiation notifie que le résultat d'exécution d'une activité est répudié ; l'exécution de l'activité correspondante est annulée.

De cette façon la sûreté de fonctionnement de l'application est assurée.

5.5.3a Résultat

La validation expérimentale de notre canevas a démontré qu'en fonction d'un plan de coordination sécurisée, nous sommes arrivés à détecter en temps réel :

- les activités qui ne sont pas exécutées par des services prévus.
- les activités qui ne sont pas exécutées avec des paramètres corrects

Nous sommes arrivés à détecter a posteriori les activités qui n'ont pas été exécutées en bon ordre ou à bon moment.

Du point de vue de l'utilisation du modèle MEO, la validation expérimentale a bien montré la contribution de ce modèle. Nous pouvons décrire le plan de coordination non sécurisée en utilisant des prédicats fournis d'ordonnancement, de déclenchement et d'inter-dépendance de données. Nous pouvons de plus exprimer les aspects sécurité de ce plan de coordination en utilisant des prédicats fournis de ces aspects.

Ces plans sont exécutés conformément au modèle d'exécution associé au modèle MEO, en tenant compte des propriétés d'exécution par défaut. Les évaluateurs implantant les algorithmes d'évaluation fournis par ce modèle d'exécution permettent d'évaluer les contraintes sur les portées spécifiques (spécialisé, local et global).

Du point de vue de l'utilisation du canevas, ses interfaces et son implantation sont utiles pour l'implantation de l'application de réservation de vols. Deux moteurs (deux configurations de l'architecture proposée) permettent de tourner ces plans conformément au modèle d'exécution spécifié. Ces configurations se différencient sur des évaluateurs dédiés aux contraintes spécialisées de sécurité et les contrôleurs de sécurité correspondants. Dans cette validation expérimentale, nous n'avons pas dû réimplanter les interfaces de MEOBI (ajouter des nouveaux codes) pour l'instanciation des moteurs qui exécutent ces deux plans. Les codes ajoutés portent sur l'interface graphique permettant de visualiser l'exécution de ces plans.

Dans le cas où les services utilisés seraient implantés par différentes formes comme agent, composant ou service Web, il s'agit d'implanter ou d'utiliser des contrôleurs d'interaction (invocateurs, connecteurs) qui confèrent à l'interopérabilité entre les composants de la configuration et les services utilisés. Par exemple, pour appeler un service Web, il faut instancier un invocateur permettant d'emballer des paramètres d'appel sous forme des messages SOAP. L'architecture de MEOBI permet d'isoler les comportements des contrôleurs d'interaction et les comportements d'autres composants du moteur. De même, pour la sécurité du moteur d'exécution lui-même, il s'agit de réutiliser des fonctionnements de sécurité au niveau de services utilisés et au niveau de l'infrastructure de communication (approche parcellaire, voir la section 2.1.2) ; cela revient à l'exploitation des services par des appels à leurs fonctions.

5.6 Conclusion

Dans ce chapitre, nous avons présenté MEOBI, un canevas de coordination sécurisée. Notre architecture permet d'exécuter un plan de coordination sécurisée d'activités par une configuration spécifique des contrôleurs, des évaluateurs et des services utilisés. D'une part, elle permet d'adapter ces composants eux-mêmes. D'autre part, notre architecture permet d'intégrer des nouveaux composants pour l'évaluation des différentes contraintes (spécialisées, locales ou globales). L'expérimentation que nous avons menée est incomplète. Nous avons néanmoins des spécifications complètes pouvant être développées sur les aspects suivants :

- La mise en œuvre des contraintes globales implicites ;
- La validation des états d'exécution du plan de coordination sécurisée d'activités.

MEOBI est similaire à certaines solutions qui visent à sécuriser des applications à base de services Web [wss, wsp06, wss07, wsf06] sur les points suivants :

Il laisse les développeurs de l'application choisir des mesures de sécurité.

Il permet de séparer les mesures répondant aux besoins applicatifs et celles qui répondent aux besoins de sécurité.

MEOBI diffère de ces solutions sur les points suivants :

a Les solutions actuelles permettent d'établir des mécanismes de sécurité qui sont orientés vers l'inter-opérabilité entre les services. Ces mécanismes se ramènent aux mécanismes de sécurité au niveau des services et au niveau de la communication. Ces solutions sont donc ad-hoc, elles définissent des mécanismes de sécurité concrets pour des standards concrets. Ces solutions ne permettent pas à un développeur de (re)considérer les besoins applicatifs au moment d'établir les mécanismes de sécurité.

b Notre solution est hybride : ad-hoc pour des contrôleurs (mécanismes de sécurité concret pour des standards concrets) et générique pour des évaluateurs (algorithme d'évaluation générique). Elle permet à un développeur de spécifier de manière séparée et uniforme les

différents aspects de l'exécution coordonnée des services, particulièrement coordination et sécurité. Cette solution permet à un développeur de (re)considérer les besoins applicatifs au moment d'établir les mécanismes de sécurité.

Chapitre 6

Conclusion

Cette thèse a présenté notre proposition pour la coordination sécurisée de services. L'approche proposée se base sur la définition de MEO, un modèle de coordination sécurisée à base des contraintes (chapitre 3), d'un modèle d'exécution de la coordination sécurisée (chapitre 4) et du canevas MEOBI pour la construction des applications à base de services (chapitre 5). Les aspects considérés de l'exécution coordonnée de services sont :

- la coordination : ordonnancement, déclenchement d'activités, inter-dépendance de données
- la sécurité : authentification, autorisation, intégrité, non-répudiation
- la sûreté de fonctionnement : en fonction de la satisfaction des contraintes et du traitement des exceptions
- l'adaptation : lors de la spécification, de la transformation et de l'exécution d'un plan de coordination sécurisée d'activités.

Nous pouvons à présent adresser un bilan de cette thèse et évoquer ses perspectives.

6.1 Bilan de travaux effectués et résultats

Étant donné l'objectif d'étudier la coordination sécurisée des services, nous avons commencé ce travail par une étude de l'état de l'art. Cette étude a couvert aussi bien les aspects coordination que les aspects sécurité des applications à base de services. Elle nous a permis de comprendre comment construire des applications à base de services et comment utiliser les mesures de sécurité existantes dans le contexte de la construction de ce type d'application. Nous avons caractérisé les aspects sécurité selon la construction des applications à base de services et à trois niveaux :

- Services : les services faisant partie de l'exécution du plan de coordination (services appelants, services appelés, tiers de confiance) ;
- Communication : l'infrastructure de communication, surtout les services d'interaction dédiés, permettant de faire communiquer les services ;
- Coordination : le plan de coordination.

A notre connaissance, aucun des travaux existants ne propose de considérer les mesures de sécurité au niveau du plan de coordination. Ce constat est à l'origine de la solution proposée dans cette thèse permettant de définir une coordination sécurisée, où nous pouvons considérer les aspects fonctionnels de coordination et les aspects non fonctionnels de sécurité du plan de coordination.

Dans notre approche [Vu06a, Vu06b], nous tenons compte des informations définissant le plan de coordination sécurisée et des services qui manipulent ces informations pour exécuter le plan. Grâce aux contraintes que nous avons définies et à ces informations (que nous pouvons récupérer

de manière sécurisée conformément aux contraintes de sécurité), nous pouvons sécuriser l'exécution du plan. Pour cela nous avons proposé un modèle de coordination sécurisée [VCVS06a], des algorithmes d'exécution, une architecture logicielle du canevas et une validation partielle du canevas [VCVS06b, VCCVS06].

6.1.1 Modèle de coordination sécurisée MEO

Nous avons proposé d'un modèle de coordination sécurisée à base de contraintes. Les activités d'un plan sont régies par différents types de contraintes : locale, globale, spécialisée. Ainsi, les aspects fonctionnels de coordination et non-fonctionnels de sécurité sont exprimés par des contraintes. De cette façon, nous pouvons considérer autant d'aspects fonctionnels et non-fonctionnels (adaptation, sûreté de fonctionnement) que nous voulons. Le système de types et les prédicats prédéfinis pour chaque aspect permettent de rendre évaluables des contraintes.

L'exécution d'un plan correspond à l'exécution des activités. L'exécution de chaque activité correspond à l'évaluation de ses contraintes locales et de son appel.

6.1.2 Architecture du MEOBI

Les composants de l'architecture sont des services. Nous distinguons des évaluateurs (globaux, locaux, spécialisés, d'appel), des contrôleurs (de coordination, de sécurité, de sûreté de fonctionnement) et des services utilisés (services à coordonner, tiers de confiance). Le protocole est défini pour la communication entre ces composants. Cette architecture s'adapte aux plans de coordination sécurisée d'activités définis par le modèle MEO. Elle exécute, contrôle et gère un plan en assurant son adaptation et sa sûreté de fonctionnement.

Un prototype du canevas a été réalisé. Ce prototype permet de démontrer les scénarios correspondant à un plan de coordination et à un plan de coordination sécurisée d'une application de réservation de vols.

6.2 Perspectives

Le prototype développé a montré la faisabilité de notre approche et de notre canevas MEOBI. Il a montré également la flexibilité du contrôle des activités d'un plan par les contraintes. Nous voulons consolider la validité de notre proposition de manière générale. Pour cela, nous prévoyons plusieurs améliorations et extensions suivantes :

1. Le modèle MEO permet de spécifier différents aspects de la coordination et de la sécurité. Nous pensons qu'il est possible d'utiliser MEO pour spécifier les autres aspects tels que la transaction, l'évolution, la tolérance aux pannes et la persistance. La recherche consiste donc à définir des propriétés et des prédicats pour ces aspects. Il est également possible d'étendre le canevas MEOBI pour les implanter : la transaction est implantée par des contrôleurs d'interaction et des évaluateurs dédiés, l'évolution est implantée par les contrôleurs d'adaptation et des évaluateurs dédiés, la tolérance aux pannes et la persistance sont implantées sous forme des contrôleurs de sûreté de fonctionnement et des évaluateurs dédiés. Cela donne une interaction de notre travail avec d'autres travaux de notre équipe qui portent sur : la composition avec des propriétés transactionnelles et persistantes (thèse en cours [Pa07]), la composition évolutive (thèse en cours [PTa08]), la coordination autonomique de services de données (thèse en cours [MGZMVS05]) et le service de tolérance aux pannes [DCC04].

2. Nous pensons aussi à une meilleur exploitation des journaux de coordination. Actuellement, les journaux de coordination servent à la validité a posteriori de l'exécution du plan, ainsi que pour l'évaluation des contraintes de non-répudiation. Ces journaux de coordination peuvent servir aux futures analyses concernant d'autres aspects non-fonctionnels. Par exemple, pour l'évolution des services faisant partie de l'exécution du plan, l'extraction des informations archivées dans les journaux de coordination permet de déterminer les services pertinents pour le remplacement des services en panne lors de l'exécution du plan. Ou bien, concernant l'aspect tolérance aux pannes, nous pouvons déterminer des points de reprise de l'exécution d'une activité ou d'un service à partir des journaux de coordination correspondants. Concernant l'aspect transaction, nous pouvons déterminer les transactions complètes en se basant sur les preuves d'exécution des activités. Tout cela implique la recherche d'une façon de bien gérer les informations archivées, également la recherche d'un meilleur stockage des journaux de coordination.
3. Notre approche de la coordination sécurisée des services est basée sur l'hypothèse que les services devant être coordonnés sont disponibles, accessibles, inter-opérables et éventuellement flexibles. Nous voulons poursuivre notre approche en levant ces hypothèses et ainsi valider notre canevas dans des granularités différentes de services : comportement, contexte de développement, domaine d'application, etc. Cela implique également une grande attention sur la mise en oeuvre des contrôleurs d'adaptation et des contrôleurs de sûreté de fonctionnement.
4. Il est souhaitable de développer un environnement qui facilite la manipulation des plans de coordination sécurisée d'activités : une interface graphique permettant de mieux visualiser les contraintes d'un plan, un générateur de code permettant de générer automatiquement une configuration des contrôleurs selon les contraintes globales du plan.

Bibliographie

[AH96] V. Atluri and W. Huang. An extended petri net model for supporting workflows in a multilevel secure environment. 10 :199–216, 1996.

[AP05] F. Pilhofer A. Puder, K. Römer. *Distributed Systems Architecture : A Middleware Approach*. Morgan Kaufmann, 2005.

[BFA99] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.*, 2(1) :65–104, 1999.

[BGH+95] M. Bellare, J. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. *i*KP - A Family of Secure Electronic Payment Protocols. In *1st USENIX Workshop on Electronic Commerce*, pages 89–106, 1995.

[BGH+00] M. Bellare, J. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, E. Van Herreweghen, and M. Waidner. Design, Implementation and Deployment of the *i*KP Secure Electronic Payment System. *IEEE Journal on Selected Areas in Communications*, 18(4) :611–627, 2000.

[Bib77] K. J. Biba. Integrity considerations for secure computer systems. Technical report, MITRE Corp, 1977.

[BL75] D.E. Bell and L.S. LaPadula. Secure computer systems : Unified exposition and multics interpretation. Technical report, MITRE Corp, 1975.

[BS02] P.A. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *J. Comput. Secur.*, 10(3) :241–271, 2002.

[Ca08] D. Cooper and al. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, 2008.

[Cam04] C. Camborde. *Sécuriser vos applications Internet : Messagerie, intranet, site web, e-commerce*. Dunod, 2004.

[CCF+] F. Cabrera, G. Copeland, T. Freund, W. Freund, J. Johnson, S. Joyce, C.Kaler, J. Klein, D. Langworthy, M. Little, F. Leymann, A. Nadalin, E. Newcomer, D. Orchard, I. Robinson, J. Sexchuck, and T. Storey. Web Services Coordination (WS-Coordination).

[CDK+02] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376, 2002.

[CG89] N. Carriero and D. Gelernter. Linda in context. *Commun. ACM*, 32(4) :444–458, 1989.

[Cha83] D. Chaum. Blind signatures for untraceable payments. *LNCS Advances in Cryptology - Crypto'82*, pages 199–203, 1983.

[cis04] Cisco Application-Oriented Networking - A Flexible Application Infrastructure. CISCO Software, 2004.

[CSV96] R. Cole, D. Shur, and C. Villamizar. IP over ATM : A Framework Document. RFC 1932, 1996.

[CW87] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. *IEEE Symposium on Security and Privacy, 1987*, 1987.

[DCC04] P-Q. Duong, E. Pérez Cortés, and C. Collet. La tolérance aux fautes adaptable pour les système à composants. *Numéro spécial de Technique et Science Informatiques (TSI), sur le thème « Système à composants adaptables et extensibles »* , 23 :205–230, 2004.

[Den76] D.E. Denning. A lattice model of secure information flow. *Comm. ACM*, 19(5) :236–243, 1976.

[DF03] R. Chandramouli D.F. Ferraiolo, D.R. Kuhn. *Role-Based Access Control*. Artech House Publishers, 2003.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22 :644–654, 1976.

[DH95] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 1883, 1995.

[dot] Microsoft .NET Framework.

[ecl] Eclipse - an open development platform.

[EDS02] S. Paraboschi E. Damiani, S. De Capitani di Vimercati and P. Samarati. Securing SOAP e-services. *International Journal of Information Security*, 2002.

[eth] Telecommunications and information exchange between systems - LAN/MAN CSMA/CD (Ethernet) Access Method and Physical Layer Specifications. IEEE 802.3.

[Fer97] J. Ferber. *Les systèmes multi-agents Vers une intelligence collective*. InterEditions, 1997.

[FGM+06] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616b, 2006.

[fip00] Digital Signature Standard (DSS). NIST FIPS 186, 2000.

[fip01] Advanced Encryption Standard. NIST FIPS 197, 2001.

[FLZ06] R. Focardi, R. Lucchi, and G. Zavattaro. Secure shared data-space coordination languages : a process algebraic surveys. *Sci. Comput. Program.*, 63(1) :3–15, 2006.

[Gla97] H.M. Gladney. Access control for large collections. *ACM Transactions on Information System*, 15(2) :154–194, 1997.

[GM06] P. Pietzuch G. Mühl, L. Fiege. *Distributed Event-Based Systems*. Springer, 2006.

[HK03a] P. C. K. Hung and K. Karlapalem. A secure workflow model. pages 33–41, 2003.

[HK03b] P.C.K. Hung and K. Karlapalem. A secure workflow model. In *ACSW Frontiers '03 : Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, pages 33–41, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[HKR05] J. Haller, Y. Karabulut, and P. Robinson. Security controls in collaborative business process. *IFIP International Federation for Information Processing*, page 247, 2005.

[HLP05] W.J. Van Den Heuvel, K. Leune, and M. P. Papazoglou. Efsoc : A layered framework for developing secure interactions between web-services. *Distrib. Parallel Databases*, 18(2) :115–145, 2005.

[HR78] M.H. Harrison and W.L. Ruzzo. Monotonie protection systems. *Foundations of Secure Computations*, 1978.

[iio04] CORBA/IIOP Specification. OMG, 2004.

[jav] Java Platform, Enterprise Edition.

[jdb] Java Database Connectivity.

[JSSB97] Sushil Jajodia, Pierangela Samarati, V. S. Subrahmanian, and Eliza Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD '97 : Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 474–485, New York, NY, USA, 1997. ACM.

[ker02] Kerberos : The Network Authentication Protocol. Massachusetts Institute of Technology, 2002.

[LC00] A. Levi and M.U. Caglayan. An efficient, dynamic and trust preserving public key infrastructure. *IEEE Symposium on Security and Privacy, 2000*, 2000.

[LDS+90] T.F. Lunt, D.E. Denning, R.R. Schell, M. Heckman, and W.R. Shockley. The seaview security model. *IEEE Transaction on Software Engineering*, 1990.

[LLL06] Y. Lee, S. Lee, and H. Lee. Development of secure event service for ubiquitous computing. *LNCS International Conference on Intelligent Computing, ICIC 2006*, 344/2006, 2006.

[MGZMVS05] D. Moreno-García, J.-L. Zechinelli-Martini, and G. Vargas-Solar. Towards an Event Management and Composition Framework for Building Adaptative Systems. In *Proceedings of the Encuentro Internacional de Computación (ENC'05)*, Puebla, México, september 2005. IEEE.

[MR99] Olivier Markowitch and Yves Roggeman. Probabilistic non-repudiation without trusted third party. 1999.

[MyS] MySQL. http ://solutions.mysql.com/ ?setlang=en.

[OPTD01] D. O'Mahony, M.A. Peirce, H. Tewari, and O'M. Donal. *Electronic payment systems for E-Commerce*. Artech House, Inc., Norwood, MA, USA, 2001.

[ora] ORACLE Database.

[owl04] OWL Web Ontology Language Reference. W3C Specification, 2004.

[Pa07] A. Portilla and al. Contract Based Behavior Model for Services Coordination. In *WEBIST*, 2007.

[Pet81] J.L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[PR85] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959, 1985.

[PTa08] A. Portilla, H. Tan, and al. Building reliable mobile services based applications. In *the 24th International Conference on Data Engineering Workshops (ICDE)*, 2008.

[Ra00] R. Rivest and al. The RC6 Block Cipher. NIST Standard, 2000.

[rfca] Internet Security Glossary, Version 2. RFC Specification.

[rfcb] Secure Electronic Transaction (SET) Supplement for the v1.0 Internet Open Trading Protocol (IOTP). RFC Specification.

[rfcc] The Transport Layer Security (TLS) Protocol Version 1.1. RFC Standard.

[rfc93] MIME (Multipurpose Internet Mail Extensions) Part One : Mechanisms for Specifying and Describing the Format of Internet Message Bodies. RFC 1521, 1993.

[rfc99a] Internet X.509 Public Key Infrastructure Certificate and CRL Profile. RFC 2459, 1999.

[rfc99b] The Secure HyperText Transfer Protocol. RFC 2660, 1999.

[rfc02] (Extensible Markup Language) XML-Signature Syntax and Processing. W3C Recommendation, 2002.

[rfc03] Wrapping a Hashed Message Authentication Code (HMAC) key with a Triple-Data Encryption Standard (DES) Key or an Advanced Encryption Standard (AES) Key. RFC 3537, 2003.

[rfc04] Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification. RFC 3851, 2004.

[Riv92] R. Rivest. The MD5 Message-Digest Algorithm, 1992.

[Riv02] R. Rivest. RSA Cryptography Standard. Public-Key Cryphtography Standard, 2002.

[RK07] P. Chodavarapu R. Kanneganti. *SOA Security*. Manning Publications Co., 2007.

[rpc88] RPC : Remote Procedure Call - Protocol Specification. RFC 1050, 1988.

[RZN+05] T. Ryutov, L. Zhou, C. Neuman, T. Leithead, and K. E. Seamons. Adaptive trust negotiation and access control. In *SACMAT '05 : Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 139–146, New York, NY, USA, 2005. ACM.

[sam05] Security Assertion Markup Language (SAML) v2.0. Oasis Specification, 2005.

[SB03] T.J. Smith and G. T. Byrd. Yalta : A Dynamic PKI and Secure Tuplespaces for Distributed Coalitions. pages 52–54, 2003.

[set97] SET Secure Electronic Transaction Specification - Version 1.0. LLC Specification, 1997.

[SFY96] R. Sandhu, C.L. Feinstein, and C.E. Youman. Role-based access control models. *IEEE Computer Magazine*, pages 38–47, 1996.

[sha04] Information technology - Security techniques - Hash-functions - Part 3 : Dedicated hash-functions. ANSI Standard, 2004.

[SK02] T. Suzuki and R. Katz. An Authorization Control Framework to Enable Service Composition Across Domains. *ACM WWW2002*, 2002.

[soa06] SOAP Version 1.2 Part 1 : Messaging Framework (Second Edition). W3C Specification, 2006.

[spk99] SPKI Requirements. RFC 2692, 1999.

[SS95] P. Smith and C. Skinner. A public-key cryptosystem and a digital signature system based on the lucas function analogue to discrete logarithms. *LNCS Advances in Cryptology - ASIACRYPT'94*, 917 :355–364, 1995.

[ssl96] SSL 3.0 Specification. Nescape Specification, 1996.

[SV00] P. Sewell and J. Vitek. Secure composition of untrusted code : wrappers and causality types. *IEEE Computer Security Foundations Workshop,*, 2000.

[tcp81] Transmission Control Protocol. RFC 793, 1981.

[TS97] R.K. Thomas and R.S. Sandhu. Task-based Authorization Controls (TBAC) : A Family of Models for Active and Enterprise-oriented Authorization Management. *IFIP WG11.3 Workshop on Database Security*, 1997.

[VCCVS06] T-H-G. Vu, C.Bobineau, C. Collet, and G. Vargas-Solar. MEOBI : Secure service coordination framework. Poster of the 22th Conference on Advanced Databases, 2006.

[VCVS06a] T-H-G. Vu, C. Collet, and G. Vargas-Solar. Defining and modeling secure service-based systems. In *Proceedings of the 14th International Conference on Cooperative Information Systems, OTM Federated Conferences and Workshops*, Montpellier, France, 2006. Springer-Verlag.

[VCVS06b] T-H-G. Vu, C. Collet, and G. Vargas-Solar. Secros : Secure service coordination. In *Proceedings of the 22th Conference on Advanced Databases*, Lille, France, 2006.

[Vu06a] T-H-G. Vu. Towards a secure service coordination. In *Workshops' Proceeding of 10th International Conference on Extending Database Technology*, pages 102–107, Munich, Germany, 2006.

[Vu06b] T-H-G. Vu. Towards a secure service coordination. In *Current Trends in Database Technology - EDBT 2006*, pages 105–114, Munich, Germany, 2006. Springer-Verlag Berlin Heidelberg.

[WO99] P. Wing and B. O'Higgins. Using Public-Key Infrastructures for Security and Risk Management. *IEEE Communication Magazine*, 1999.

[wsa04] Web Services Addressing (WS-Addressing). W3C Standard, 2004.

[wsb07] Web Services Business Process Execution Language Version 2.0. OASIS Standard, 2007.

[wsc02] Web Service Choreography Interface (WSCI) 1.0. W3C Specification, 2002.

[wsc05] Web Services Choreography Description Language Version 1.0. W3C Specification, 2005.

[wsf06] Web Services Federation Language (WS-Federation). W3C Standard, 2006.

[wsp06] Web Services Policy 1.2 - Framework (WS-Policy). W3C Standard, 2006.

[wss] Web service security. OASIS Standard.

[wss07] WS-SecureConversation 1.3. OASIS Standard, 2007.

[wst00] Web Services Trust Language. W3C Standard, 2000.

[x9598] Triple Data Encryption Algorithm Modes of Operation. ANSI X9.52 :1998, 1998.

[xml02] XML Encryption Syntax and Processing. W3C Recommendation, 2002.

[xpd05] Workflow Standard : Process Definition Interface – XML Process Definition Language. The Workflow Management Coalition Standard, 2005.

[YWS03] T. Yu, M. Winslett, and K.E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 6(1) :1–42, 2003.

[Zho01] J. Zhou. *Non-repudiation in electronic commerce.* Artech House, Inc., Norwood, MA, USA, 2001.

[ZOL05] J. Zhou, J. Onieva, and J. Lopez. Optimized multi-party certified email protocols. *Information Management and Computer Security*, 13(5) :350 – 366, 2005.